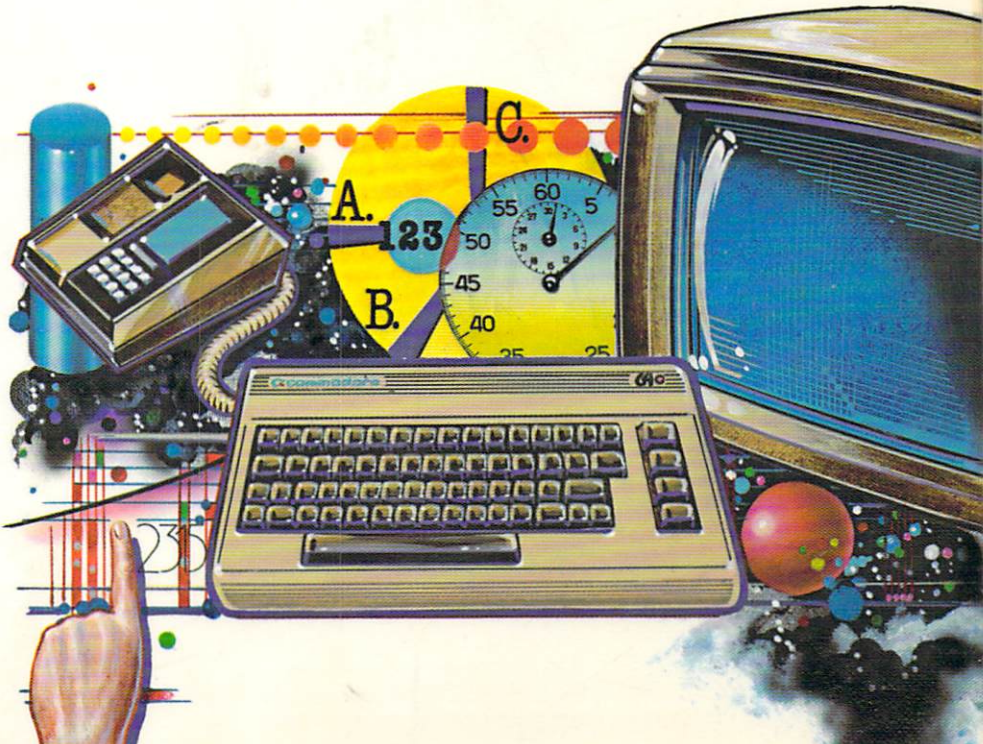
More Than 32 Basic Programs For The Commodore 64 Computer



Tom Rugg Phil Feldman & Western Systems Group

More than 32 BASIC Programs for the Commodore 64® Computer

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Rugg, Tom.

More than 32 BASIC programs for the Commodore 64 computer.

Bibliography: p.

Includes index.

1. Commodore 64 (Computer)—Programming. 2. Basic (Computer program language). I. Feldman, Phil. II. Western Systems Group. III. Title. IV. Title: More than thirty-two BASIC programs for the Commodore 64 computer.

QA76.8.C64R84 1983 001.64'25 83-5218
ISBN 0-88056-112-2

Printed in the United States of America

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005

Acknowledgements

Our thanks to Merl Miller and his staff for their help and encouragement.

AN IMPORTANT NOTE

The publisher and authors have made every effort to assure that the computer programs are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

Preface

You have bought yourself a Commodore 64 Computer (or maybe you just have access to one at school or work). You will soon find that the most frequent question you are asked goes something like this: "Oh, you got a computer, eh? Uh...what are you going to do with it?"

Your answer, of course, depends on your own particular situation. Maybe you got it for mathematical work, or for your business, or for home usage, or to enable you to learn more about computers. Maybe you got it for a teaching/learning tool or for playing games.

Even if you got the computer specifically for only one of these reasons, you should not neglect the others. The computer is such a powerful tool that it can be used in many different ways. If it is not being used for its "intended" function right now, why not make use of it in some other way?

The Commodore 64 is so small and portable that you can, say, take it home from work over the weekend and let the kids play educational games. They will have fun *and* learn a lot. After they go to bed, you can use it to help plan your personal finances. Or, you can let your guests at a party try to outsmart the computer (or each other) at some fascinating games. The possibilities go on and on.

All these things can be done with the Commodore 64, but it cannot do any of them without the key ingredient—a computer program. People with little or no exposure to computers may be in for a surprise when they learn this. A computer without a program is like a car without a driver. It just sits there.

So you ask, "Where can I get some programs to do the things I want my computer to do?" Glad you asked. There are several alternatives.

1. Hire a computer programmer. If you have a big budget, this is the way to go. Good programmers are expensive and hard to find (and you will not know for sure if they're really good until after the job is finished). Writing a couple of programs that are moderately complex will probably cost you more than you paid for the computer itself.
2. Learn to program yourself. This is a nice alternative, but it takes time. There are lots of programming books available—some are good, some are not so good. You can take courses at local colleges. If you can afford the time and you have a fair amount of common sense and inner drive, this is a good solution.
3. Buy the programs you want. This is cheaper than hiring your own programmer because all the buyers share the cost of writing the programs. You still will not find it very cheap, especially if you want to accumulate several dozen programs. Each program might cost anywhere from a few dollars to several hundred dollars. The main problem is that you cannot be sure how good the programs are, and, since they are generalized for all possible buyers, you may not be able to easily modify them to do exactly what *you* want. Also, they have to be written in a computer language that *your* computer understands. Even if you find a program written in the BASIC language, you will soon learn that Commodore BASIC is not the same as other versions. Variations between versions of the same language typically result in the program not working.

This book gives you the chance to take the third alternative at the lowest possible cost. If you divide the cost of the book by the number of programs in it (use your computer if you like), you will find that the cost per program is amazingly low. Even if there are only a few programs in the book that will be useful to you, the cost is pretty hard to beat.

Just as important is the fact that these programs are written specifically for your Commodore 64. If you type them in exactly as shown, they will work! No changes are needed. In addition,

we show you exactly what to change in order to make some simple modifications that may suit your taste or needs. Plus, if you have learned a little about BASIC, you can go even further and follow the suggestions about more extensive changes that can be made. This approach was used to try to make every program useful to you, whether you are a total beginner or an old hand with computers.

But enough of the sales pitch. Our main point is that we feel a computer is an incredibly flexible machine, and it is a shame to put it to only one or two limited uses and let it sit idle the rest of the time. We are giving you a pretty wide range of things to do with your computer, and we are really only scratching the surface.

So open your eyes and your mind! Play a mental game against the computer (WARI, JOT). Evaluate your next financial decision (LOAN, DECIDE). Expand your vocabulary or improve your reading speed (VOCAB, TACHIST). Solve mathematical equations (DIFFEQN, SIMEQN).

But please, don't leave your computer asleep in the corner too much. Give it some exercise.

How to Use This Book

Each chapter of this book presents a computer program that runs on a Commodore 64 Computer. Each chapter is made up of eight sections that serve the following functions:

1. **Purpose:** Explains what the program does and why you might want to use it.
2. **How To Use It:** Gives the details of what happens when you run the program. Explains your options and the meanings of any responses you might give. Provides details of any limitations of the program or errors that might occur.
3. **Sample Run:** Shows you what you will see on the screen when you run the program.
4. **Program Listing:** Provides a "listing" (or "print-out") of the BASIC program. These are the instructions to the computer that you must provide so it will know what to do. You must type them in extremely carefully for correct results.
5. **Easy Changes:** Shows you some very simple changes you can make to the program to cause it to work differently, if you wish. You do not have to understand how to program to make these changes.
6. **Main Routines:** Explains the general logic of the program, in case you want to figure out how it works. Gives the BASIC line numbers and a brief explanation of what each major portion of the program accomplishes.
7. **Main Variables:** Explains what each of the key variables in the program is used for, in case you want to figure out how it works.

8. **Suggested Projects:** Provides a few ideas for major changes you might want to make to the program. To try any of these, you will need to understand BASIC and use the information provided in the previous two sections (Main Routines and Main Variables).

To use any of these programs on your Commodore 64, you need only use the first four sections. The last four sections are there to give you supplementary information if you want to tinker with the program.

RECOMMENDED PROCEDURE

Here is our recommendation of how to try any of the programs in this book:

1. Read through the documentation that came with the Commodore 64 to learn the fundamentals of communication with the computer. This will teach you how to turn the computer on, enter a program, correct mistakes, run a program, etc.
2. Pick a chapter and read Section 1 ("Purpose") to see if the program sounds interesting or useful to you. If not, move on to the next chapter until you find one that is. If you are a beginner you might want to try one of the short "Miscellaneous Programs" first.
3. Read Sections 2 and 3 of the chapter ("How To Use It" and "Sample Run") to learn the details of what the program does.
4. Enter the NEW command to eliminate any existing program that might already be in your computer's memory. Using Section 4 of the chapter ("Program Listing"), *carefully* enter the program into the computer. Be particularly careful to get all the punctuation characters right (i.e., commas, semicolons, colons, quotation marks, etc.).
5. After the entire program is entered into the computer's memory, use the LIST command to display what you have entered so you can double check for typographical errors, omitted lines, etc. Don't mistake a semicolon for a colon, or an alphabetic I or O for a numeric 1 or 0 (zero). *Take a minute to note the differences in these characters before you begin.*

6. Before trying to RUN the program, use the SAVE "name" command to save the program temporarily on cassette. This could prevent a lot of wasted effort in case something goes wrong (power failure, computer malfunction, etc.). If the computer "hangs up" when you enter RUN, you can simply reset it, reload the program from cassette, and look for typing errors.
7. Now RUN the program. Is the same thing happening that is shown in the Sample Run? If so, accept our congratulations and go on to step 9. If not, stay cool and go to step 8.
8. If you got a SYNTAX ERROR in a line, LIST that line and look at it closely. Something is not right. Maybe you interchanged a colon and a semicolon. Maybe you typed a numeric 1 or 0 instead of an alphabetic I or O. Maybe you misspelled a word or omitted one. Keep looking until you find it, then correct the error and go back to step 7.

   If you got some other kind of error message, consult the computer's documentation for an explanation. Keep in mind that the error might not be in the line that is pointed to by the error message. It is not unusual for the mistake to be in a line immediately preceding the error message line. Another possibility is that one or more lines were omitted entirely. In any event, fix the problem and go back to step 7.

   If there are no error messages, but the program is not doing the same thing as the Sample Run, there are two possibilities. First, maybe the program isn't *supposed* to do exactly the same thing. Some of the programs are designed to do unpredictable things to avoid repetition (primarily the game programs and graphic displays). They should be doing the same *types* of things as the Sample Run, however.

   The second possibility is that you made a typing error that did not cause an error message to be displayed, but simply changed the meaning of one or more lines in the program. This can be a little tricky to find, but you can usually narrow it down to the general area of the problem by noting the point at which the error takes place. Is the first thing displayed correct? If so, the error is probably after the PRINT statement that caused the first thing to be displayed. Look for the same types of things mentioned before. Make the corrections and go back to step 7.

9. Continue running the program, trying to duplicate the Sample Run. If you find a variation that cannot be accounted for in the "How To Use It" section of the chapter, go to step 8. Otherwise, if it seems to be running properly, SAVE the program on cassette.
10. Read Section 5 of the chapter ("Easy Changes"). Try any of the changes that look interesting. If you think the changed version is better, SAVE it on cassette, too. You will probably want to give it a slightly different title in the first REM statement to avoid future confusion.

A NOTE ON THE PROGRAM LISTINGS

A line on the screen of the Commodore 64 is 40 characters wide. However, the printer that was used to create the Program Listing section of each chapter prints lines up to 80 characters long. When typing into your computer a line longer than 40 characters, simply type the entire line as shown in the listing followed by the **RETURN** key. Don't be fooled by the fact that the cursor on your Commodore 64 jumps down to the next line after you enter the 40th character—it's just one long line until you press **RETURN**.

Contents

Section 5 – MATHEMATICS PROGRAMS

For math, engineering, and statistical uses.

Section 6 – MISCELLANEOUS PROGRAMS

Short programs that do interesting things.

Section 1

Applications Programs

INTRODUCTION TO APPLICATIONS PROGRAMS

Good practical applications are certainly a prime use of personal computers. There are a myriad of ways the Commodore 64 can help us to do useful work. Here are eight programs for use around the home or business.

Financial considerations are always important. LOAN will calculate interest, payment schedules etc. for mortgages, car loans, or any such business loan. Do you ever have trouble balancing your checkbook(s)? CHECKBOOK will enable you to rectify your monthly statements and help you find the cause of any errors. With the many types of investments available today, there is often confusion about their true annual yields. ANNUAL will make sure you don't have this problem any more.

Perhaps you find yourself compiling various lists at home or work. These could be lists of names, words, phrases, etc. The chore of alphabetizing such a list is duck soup for SORTLIST.

Fuel usage is a constant concern for those of us who drive. MILEAGE will determine and keep track of a motor vehicle's general operating efficiency.

The tedium of analyzing questionnaires and examinations can be greatly relieved with the aid of your computer. In particular, teachers and market researchers should find QUEST/EXAM useful.

Often we are faced with difficult decisions. DECIDE transforms the Commodore 64 into a trusty advisor. Help will be at hand for any decision involving the selection of one alternative from several choices.

Before anything else, you might want to consult BIORHYTHM each day. Some major airlines, and other industries, are placing credence on biorhythm theory. If you agree, or "just in case," simply turn on your computer and load this program.

ANNUAL

PURPOSE

Suppose you put $1000.00 into an investment that pays ten percent interest. How much interest will you earn by the end of one year?

Generally the answer is *not* simply $100.00 (the interest rate times the principal). The amount of interest earned per year depends on how often the interest is compounded (calculated) and paid. If it is done only once at the end of the year, you really do earn only ten percent. But if it is done more often (each month, for example) you earn more. This is because the interest after the first month begins earning interest too.

This program shows you the annual yield of any interest rate for various compounding techniques, assuming that the interest is added to the principal as it is calculated. Be aware that some savings institutions use different techniques for calculating annual yield. Their published figures may differ from the ones calculated here.

HOW TO USE IT

Simply enter the interest rate you want to evaluate. The program shows the annual yield for annual, semi-annual, quarterly, monthly, weekly, and daily compounding. Then it asks you for another interest rate to evaluate. If you have no more, enter zero to end the program.

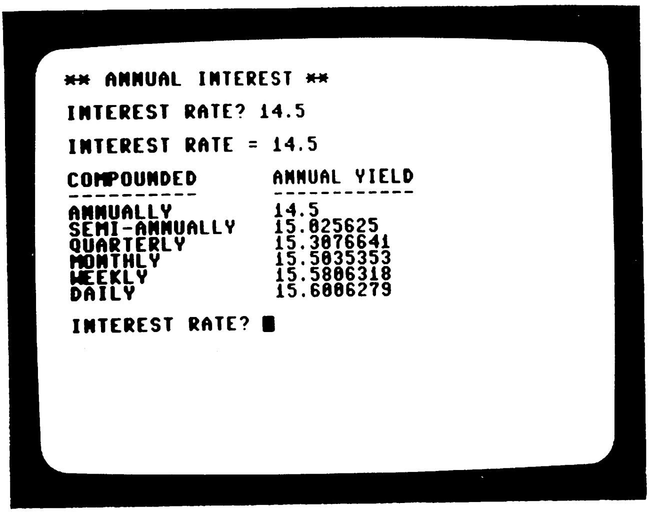
Due to the way numbers are represented inside the computer, you may occasionally notice the last significant digit to be inac-

curate. Fortunately, there is seldom any need to know the annual yeild beyond two or three decimal places, so this should be no problem.

The calculated answers come out almost instantaneously for most compounding techniques, but daily compounding takes a little longer to compute (about 3 or 4 seconds).

The Easy Changes section below shows how to send the output to a printer, if you have one. It also shows how to add other compounding techniques to the program.

SAMPLE RUN



The program asks what interest rate should be evaluated. The operator provides it, and the program shows the annual yield for six different compounding techniques.

PROGRAM LISTING

```
READY.

100 REM: ANNUAL INTEREST
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
```

```
130 CLR:PRINT CHR$(147)
150 PRINT"** ANNUAL INTEREST **"
160 PRINT
170 INPUT"INTEREST RATE";R
180 PRINT:IF R=0 THEN END
190 PRINT"INTEREST RATE =";R
200 PRINT
210 PRINT"COMPOUNDED";TAB(16);
220 PRINT"ANNUAL YIELD"
230 PRINT"----------";TAB(16);
240 PRINT"------------"
250 RESTORE
260 READ N$,N
270 IF N$ = "END" THEN 400
280 PRINT N$;TAB(15)
290 T = R/100:W = T/N:S = 1
300 FOR J = 1 TO N
310 S = S + S*W:NEXT
320 S = (S-1)*100
330 PRINT S
340 GOTO 260
400 PRINT
410 GOTO 170
800 END
900 DATA ANNUALLY     ,1
910 DATA SEMI-ANNUALLY,2
920 DATA QUARTERLY    ,4
930 DATA MONTHLY      ,12
940 DATA WEEKLY       ,52
950 DATA DAILY        ,365
960 DATA END,999

READY.
```

EASY CHANGES

1. If you have a printer, you can send the output to it very easily. Just add these lines:

   185 OPEN 1,4:CMD 1
   405 PRINT#1:CLOSE 1

2. Adding another compounding technique is done by inserting another DATA statement between lines 900-950. For example, bimonthly compounding would mean once every two months, or six times per year. To include it, add this statement:

   925 DATA BIMONTHLY,6

3. The program can be changed to make calculations for a whole series of interest rates without stopping. If you have a printer, you may want to create your own reference tables. To do so for, say, interest rates from 10 to 12 percent in increments of one-quarter percent, make these changes:

```
170 FOR R=10 TO 12 STEP .25
410 NEXT
```

MAIN ROUTINES

130	Initializes variables.
150-160	Displays title.
170-180	Gets interest rate. Ends program if zero.
190-240	Displays rate and column headings.
250-410	Makes calculation and displays result for each compounding technique.
900-960	DATA statements for each compounding technique.

MAIN VARIABLES

R	Interest rate supplied by operator (as percentage).
N$	Name of compounding technique.
N	Number of times per year to compound.
T	Interest rate (as decimal).
W	Interest rate per compounding period.
S	Sum of interest earned for the year.
J	Loop variable.

SUGGESTED PROJECTS

1. Take inflation and taxes into account and show the "real" gain or loss of the investment. For example, a person in a 50 percent tax bracket during a year of nine percent inflation needs to make about an 18 percent annual yield just to break even at the end of the year.
2. Change the program to display (or print) a table of annual yields. Show a column for each compounding technique, and a row for each interest rate.

BIORHYTHM

PURPOSE

Did you ever have one of those days when nothing seemed to go right? All of us seem to have days when we are clumsy, feel depressed, or just cannot seem to force ourselves to concentrate as well as usual. Sometimes we know why this occurs. It may result from the onset of a cold or because of an argument with a relative. Sometimes, however, we find no such reason. Why can't we perform up to par on some of those days when nothing is known to be wrong?

Biorhythm theory says that all of us have cycles, beginning with the moment of birth, that influence our physical, emotional, and intellectual states. We will not go into a lot of detail about how biorhythm theory was developed (your local library probably has some books about this if you want to find out more), but we will summarize how it supposedly affects you.

The physical cycle is twenty-three days long. For the first 11½ days, you are in the positive half of the cycle. This means you should have a feeling of physical well-being, strength, and endurance. During the second 11½ days, you are in the negative half of the cycle. This results in less endurance and a tendency toward a general feeling of fatigue.

The emotional cycle lasts for twenty-eight days. During the positive half (the first fourteen days), you should feel more cheerful, optimistic, and cooperative. During the negative half, you will tend to be more moody, pessimistic, and irritable.

The third cycle is the intellectual cycle, which lasts for thirty-three days. The first half is a period in which you should have

greater success in learning new material and pursuing creative, intellectual activities. During the second half, you are supposedly better off reviewing old material rather than attempting to learn difficult new concepts.

The ups and downs of these cycles are relative to each individual. For example, if you are a very self-controlled, unemotional person to begin with, your emotional highs and lows may not be very noticeable. Similarly, your physical and intellectual fluctuations depend upon your physical condition and intellectual capacity.

The day that any of these three cycles changes from the plus side to the minus side (or vice versa) is called a "critical day." Biorhythm theory says that you are more accident-prone on critical days in your physical or emotional cycles. Critical days in the intellectual cycle aren't considered as dangerous, but if they coincide with a critical day in one of the other cycles, the potential problem can increase. As you might expect, a triple critical day is one on which you are recommended to be especially careful.

Please note that there is quite a bit of controversy about biorhythms. Most scientists feel that there is not nearly enough evidence to conclude that biorhythms can tell you anything meaningful. Others believe that biorhythm cycles exist, but that they are not as simple and inflexible as the 23, 28, and 33 day cycles mentioned here.

Whether biorhythms are good, bad, true, false, or anything else is not our concern here. We are just presenting the idea to you as an interesting theory that you can investigate with the help of your Commodore 64.

HOW TO USE IT

The program first asks for the birth date of the person whose biorhythm cycles are to be charted. You provide the month and day as you might expect. For the year, you only need to enter the last two digits if it is between 1900 and 1999. Otherwise, enter all four digits.

Next the program asks you for the start date for the biorhythm chart. Enter it in the same way. Of course, this date cannot be earlier than the birth date.

After a delay of about a second, the program clears the screen and begins plotting the biorhythm chart, one day at a time. The left side of the screen displays the date, while the right side

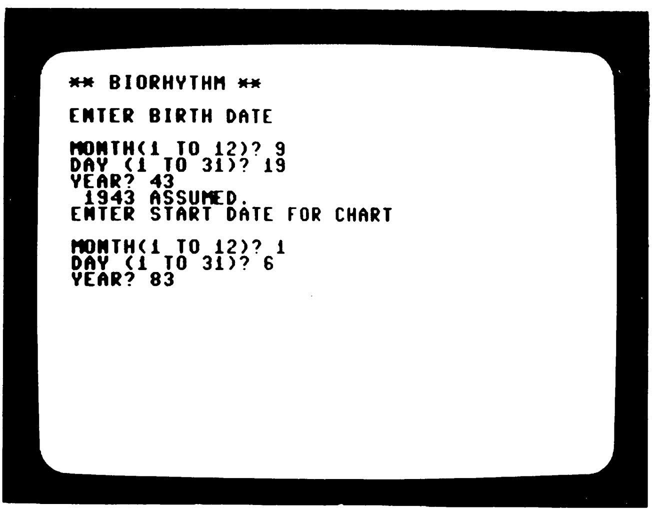
displays the chart. The left half of the chart is the "down" (negative) side of each cycle. The right half is the "up" (positive) side. The center line shows the critical days when you are at a zero point (neither positive nor negative).

Each of the three curves is plotted with an identifying letter—P for physical, E for emotional, and I for intellectual. When the curves cross, an asterisk is displayed instead of either of the two (or three) letters.

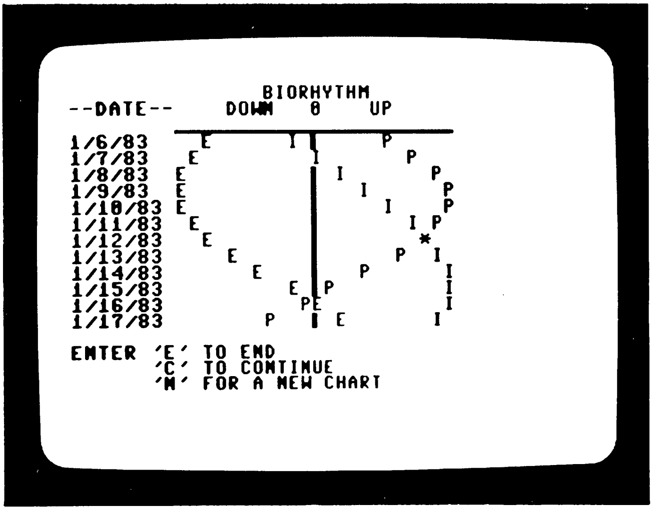
Twelve days of the chart are displayed on one screen, and then the program waits for you to press a key. If you press the N key, the current chart ends and the program starts over again. If you press the C key the program clears the screen and displays the next twelve days of the chart. If you press the E key, the program ends.

The program will allow you to enter dates from the year 100 A.D. and on. We make no guarantees about any extreme future dates, however, such as entering a year greater than 3000. We sincerely hope that these limitations do not prove to be too confining for you.

SAMPLE RUN



The operator enters his or her birth date and the date for the beginning of the chart.



The program responds with the first 12 days of the operator's biorhythm chart, then requests continuation options.

PROGRAM LISTING

```
READY.

100 REM: BIORHYTHM
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 CLR:PRINT CHR$(147)
130 L=0:T=11:P=3.14159265
140 PRINT"** BIORHYTHM **":PRINT
150 PRINT"ENTER BIRTH DATE"
160 GOSUB 500:GOSUB 600:JB=JD
190 PRINT"ENTER START DATE FOR CHART"
200 GOSUB 500:GOSUB 600:JC=JD
230 IF JC>=JB THEN 270
240 PRINT"CHART DATE CAN'T BE EARLIER"
250 PRINT"THAN BIRTH DATE."
260 PRINT"TRY AGAIN":PRINT:GOTO 150
270 FOR K = 1 TO 1000:NEXT
280 GOSUB 700
300 N = JC-JB
```

```
310 V = 23:GOSUB 800
320 V = 28:GOSUB 800
330 V = 33:GOSUB 800
340 GOSUB 1000
350 PRINT C$;TAB(8);L$
360 JC = JC+1:L = L+1:IF L<12 THEN 300
370 PRINT
372 PRINT"ENTER 'E' TO END"
374 PRINT"      'C' TO CONTINUE"
376 PRINT"      'N' FOR A NEW CHART"
380 GET R$:IF R$ = "" THEN 380
385 IF R$ = "N" THEN 120
390 IF R$ = "E" THEN END
395 IF R$ <> "C" THEN 370
400 L = 0:GOTO 280
500 PRINT
505 INPUT"MONTH(1 TO 12)";M
510 M = INT(M):IF M<1 OR M>12 THEN 505
520 INPUT"DAY (1 TO 31)";D
530 D = INT(D):IF D<1 OR D>31 THEN 520
540 INPUT"YEAR";Y
550 Y = INT(Y):IF Y<0 THEN 540
560 IF Y>99 THEN 580
570 Y = Y+1900:PRINT Y;"ASSUMED."
580 RETURN
600 W = 0:IF M<3 THEN W = -1
610 JD = INT(1461*(Y+4800+W)/4)
620 B = INT(367*(M-2-W*12)/12)
630 IF B<0 THEN B = B+1
640 JD = JD+B
650 B = INT(INT(3*(Y+4900+W)/100)/4)
660 JD = JD+D-32075-B
670 RETURN
700 PRINT CHR$(147)
705 POKE 53280,3:POKE 53281,7:PRINT CHR$(28)
710 PRINT TAB(15);"BIORHYTHM"
720 PRINT"--DATE--";TAB(12)
730 PRINT"DOWN";TAB(19);"0";TAB(24);"UP"
740 PRINT TAB(8)
750 FOR K = 1 TO T+T+1:PRINT CHR$(175);
760 NEXT:PRINT:RETURN
800 W = INT(N/V):R = N-(W*V)
850 IF V<>23 THEN 900
860 L$ = CHR$(32):FOR K = 1 TO 4
870 L$ = L$ + L$:NEXT
880 L$=LEFT$(L$,T)+CHR$(161)+LEFT$(L$,T)
890 IF V = 23 THEN C$ = "P"
900 IF V = 28 THEN C$ = "E"
910 IF V = 33 THEN C$ = "I"
920 W = R/V:W = W*2*P
930 W = T*SIN(W):W = W+T+1.5
```

```
940 W = INT(W):A$ = MID$(L$,W,1)
950 IF A$="P" OR A$="E" OR A$="*" THEN C$ = "*"
955 IF W = 1 THEN 980
960 L$ = LEFT$(L$,W-1)+C$+RIGHT$(L$,T+T+1-W)
970 RETURN
980 L$=C$+RIGHT$(L$,T+T):RETURN
990 L$=LEFT$(L$,T+T)+C$:RETURN
1000 W = JC+68569:R = INT(4*W/146097)
1010 W = W-INT((146097*R+3)/4)
1020 Y = INT(4000*(W+1)/1461001)
1030 W = W-INT(1461*Y/4)+31
1040 M = INT(80*W/2447)
1050 D = W-INT(2447*M/80)
1060 W = INT(M/11):M = M+2-12*W
1070 Y = 100*(R-49)+Y+W
1080 A$ = STR$(M):W = LEN(A$)-1
1090 C$ = MID$(A$,2,W)+"/"
1100 A$ = STR$(D):W = LEN(A$)-1
1110 C$ = C$+MID$(A$,2,W)+"/"
1120 A$ = STR$(Y):W = LEN(A$)-1
1130 C$ = C$+MID$(A$,W,2)
1140 RETURN

READY.
```

EASY CHANGES

1. Want to see the number of days between any two dates? Insert this line:

   305 PRINT "DAYS=";N: END

   Then enter the earlier date as the birth date, and the later date as the start date for the chart. This will cause the program to display the difference in days and then end.
2. To alter the number of days of the chart shown on each screen, alter the 12 in line 360.

MAIN ROUTINES

120- 140	Initializes variables. Displays titles.
150- 160	Asks for birth date and converts to Julian date format (i.e., the number of days since January 1, 4713 B.C.
190- 200	Asks for start date for chart and converts to Julian date format.

230- 260	Checks that chart date is not sooner than birth date.
270	Delays about one second before displaying chart.
280	Displays heading at top of screen.
300	Determines number of days between birth date and current chart date.
310- 330	Plots points in L$ string for each of the three cycles.
340	Converts Julian date back into month-day-year format.
350	Displays one line on the chart.
360- 400	Adds one to chart date. Checks to see if the screen is full.
500- 580	Subroutine to ask operator for month, day, and year. Edits replies.
600- 670	Subroutine to convert month, day, and year into Julian date format.
700- 760	Subroutine to clear screen and display headings.
800- 990	Subroutine to calculate remainder R of N/V, and plot a point in L$ based on V and R.
1000-1140	Subroutine to convert Julian date JC back into month-day-year format.

MAIN VARIABLES

L	Counter of number of lines on screen.
T	Number of characters on one side of the center of the chart.
P	Pi.
JB	Birth date in Julian format.
JD	Julian date calculated in subroutine.
JC	Chart start date in Julian format.
K	Loop and work variable.
N	Number of days between birth and current chart date.
V	Number of days in present biorhythm cycle (23, 28, or 33).
C$	String with date in month/day/year format.
L$	String with one line of the biorhythm chart.
R$	Reply from operator after screen fills up.
M	Month (1-12)
D	Day (1-31)
Y	Year (100 or greater)

W, B	Work variables.
R	Remainder of N/V (number of days into cycle).
A$	Work variable.

SUGGESTED PROJECTS

1. Investigate the biorhythms of some famous historical or athletic personalities. For example, are track and field athletes usually in the positive side of the physical cycle on the days that they set world records? Where was Lincoln in his emotional and intellectual cycles when he wrote "The Gettysburg Address"? Do a significant percentage of accidents befall people on critical days?
2. Modify the program to print the chart on a line printer. (Be sure to print the name and/or birthdate on the chart, too.)

CHECKBOOK

PURPOSE

Many people consider the monthly ritual of balancing the checkbook to be an irritating and error-prone activity. Some people get confused and simply give up after the first try, while others give up the first time they cannot reconcile the bank statement with the checkbook. Fortunately, you have an advantage—your computer. This program takes you through the necessary steps to balance your checkbook, doing the arithmetic for you, of course.

HOW TO USE IT

The program starts off by giving you instructions to verify that the amount of each check and deposit are the same on the statement as they are in your checkbook. Sometimes the bank will make an error in reading the amount that you wrote on a check (especially if your handwriting is not too clear), and sometimes you will copy the amount incorrectly into your checkbook. While you are comparing these figures, make a check mark in your checkbook next to each check and deposit listed on the statement. A good system is to alternate the marks you use each month (maybe an "x" one month and a check mark the next) so you can easily see which checks and deposits came through on which statement.

Next, the program asks for the ending balance shown on the bank statement. You are then asked for the *check number* (not the amount) of the most recent check shown on the statement. This will generally be the highest numbered check the bank has

processed, unless you like to write checks out of sequence. Your account balance after this most recent check will be reconciled with the statement balance, so that is what the program asks for next—your checkbook balance after the most recent check.

The program must compensate for any differences between what your checkbook has in it prior to the most recent check and what the statement has on it. First, if you have any deposits that are not shown on the statement before the most recent check, you must enter them. Generally, there are none, so you just enter "END."

Next you have to enter the amounts of any checks that have not yet "cleared" the bank and that are prior to the most recent check. Look in your checkbook for any checks that do not have your check mark next to them. Remember that some of these could be several months old.

Next you enter the amount of any service charges or debit memos that are on the statement, but which have not been shown in your checkbook prior to the most recent check. Typically, this is just a monthly service charge, but there might also be charges for printing new checks for you or some other adjustment that takes money away from you. Credit memos (which give money back to you) are not entered until later. Be sure to make an entry in your checkbook for any of these adjustments so that next month's statement will balance.

Finally, you are asked for any recent deposits or credit memos that were *not* entered in your checkbook prior to the most recent check, but that *are* listed on the bank statement. It is not unusual to have one or two of these, since deposits are generally processed by banks sooner than checks.

Now comes the moment of truth. The program tells you whether or not you are in balance and displays the totals. If so, pack things up until next month's statement arrives.

If not, you have to figure out what is wrong. The best thing to do first is to make sure you entered all the data correctly. You can verify that the outstanding checks were entered correctly with this command:

```
FOR J=1 TO NC:PRINT C(J);:NEXT
```

Then, to review the balancing summary, you can enter:

```
GOTO 810
```

To verify that you entered everything else correctly, simply RUN the program again and compare results.

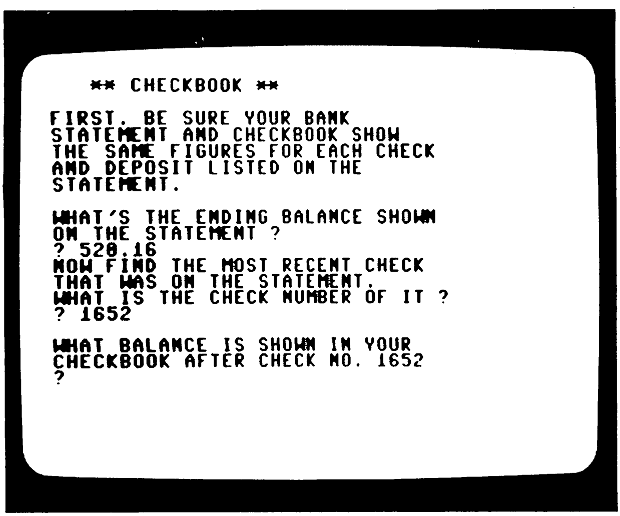
If you entered everything correctly, the most likely cause of the out of balance condition is an arithmetic error in your checkbook. Look for errors in your addition and subtraction, with subtraction being the most likely culprit. This is especially likely if the amount of the error is a nice even number like one dollar or ten cents.

Another common error is accidentally adding the amount of a check in your checkbook instead of subtracting it. If you did this, your error will be twice the amount of the check (which makes it easy to find).

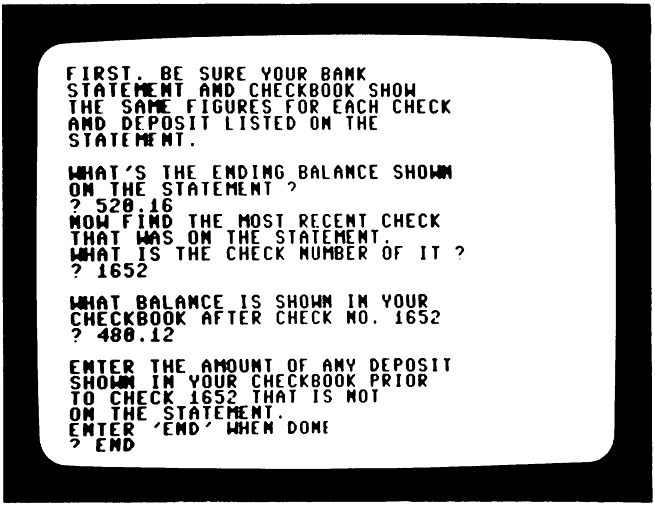
If this still does not explain the error, check to be sure you subtracted *last* month's service charge when you balanced your checkbook with the previous statement. And, of course, if you did not balance your checkbook last month, you cannot expect it to come out right this month.

The program has limitations of how many outstanding checks you can enter, but this can be changed easily. See "Easy Changes" below.

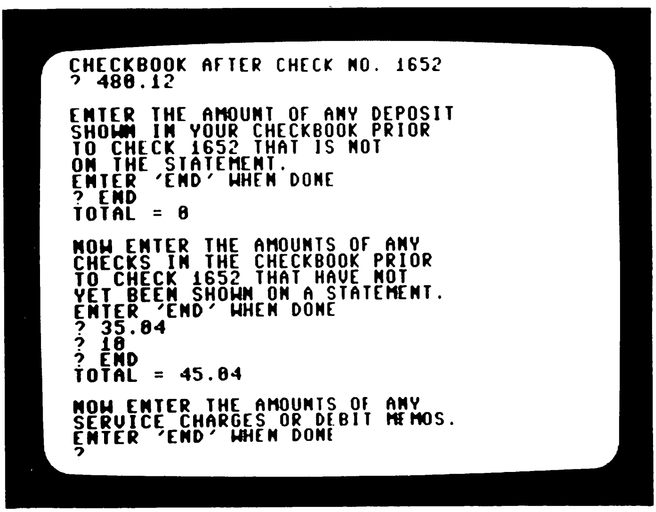
SAMPLE RUN



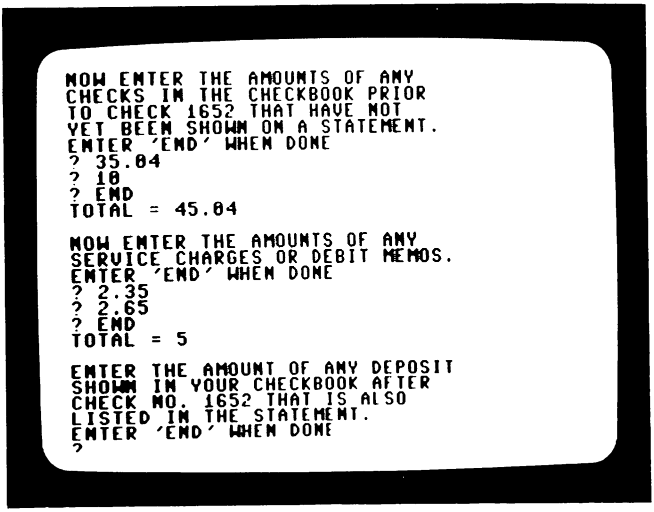
The program displays an introduction, and the operator begins providing the necessary information.



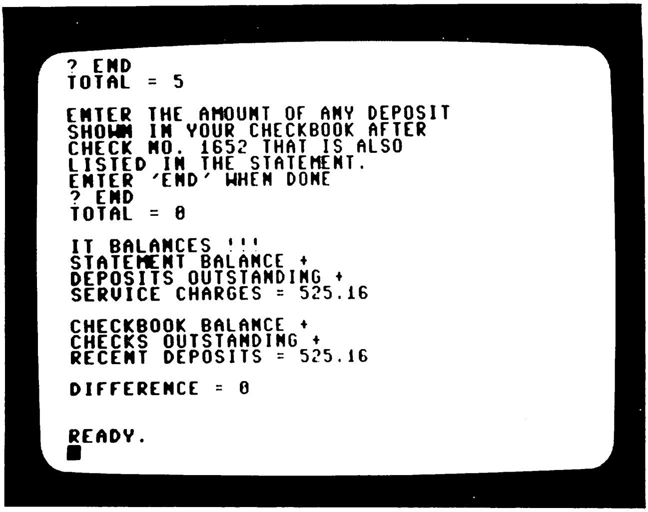
The operator continues by entering the checkbook balance, followed by END to indicate no outstanding deposits.



The operator enters the outstanding checks, and prepares to enter service charges.



After the service charges are entered, the operator will indicate no late deposits by entering END.



Finally, the program displays balancing information and ends.

PROGRAM LISTING

```
READY.

100 REM: CHECKBOOK
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 CLR:PRINT CHR$(147)
130 PRINT"    ** CHECKBOOK **":PRINT
150 MC = 10
160 DIM C(MC)
180 E$ = "ERROR. RE-ENTER."
190 PRINT"FIRST. BE SURE YOUR BANK"
200 PRINT"STATEMENT AND CHECKBOOK SHOW"
210 PRINT"THE SAME FIGURES FOR EACH CHECK"
220 PRINT"AND DEPOSIT LISTED ON THE"
230 PRINT"STATEMENT.":PRINT
280 PRINT"WHAT'S THE ENDING BALANCE SHOWN"
290 PRINT"ON THE STATEMENT ?":INPUT SB
300 PRINT"NOW FIND THE MOST RECENT CHECK"
310 PRINT"THAT WAS ON THE STATEMENT."
330 PRINT"WHAT IS THE CHECK NUMBER OF IT ?"
340 INPUT LC
350 L$ = "NO MORE ROOM."
380 PRINT
390 PRINT"WHAT BALANCE IS SHOWN IN YOUR"
400 PRINT"CHECKBOOK AFTER CHECK NO.";LC
410 INPUT CB:PRINT
430 PRINT"ENTER THE AMOUNT OF ANY DEPOSIT"
440 PRINT"SHOWN IN YOUR CHECKBOOK PRIOR"
450 PRINT"TO CHECK";LC;"THAT IS NOT"
460 PRINT"ON THE STATEMENT."
470 A$ = "ENTER 'END' WHEN DONE":PRINT A$
480 INPUT R$:IF R$ = "END" THEN 540
500 IF VAL(R$) > 0 THEN 520
510 PRINT E$:GOTO 470
520 ND = ND+1:TD = TD+VAL(R$)
530 GOTO 480
540 PRINT"TOTAL =";TD:PRINT
550 PRINT"NOW ENTER THE AMOUNTS OF ANY"
560 PRINT"CHECKS IN THE CHECKBOOK PRIOR"
570 PRINT"TO CHECK";LC;"THAT HAVE NOT"
580 PRINT"YET BEEN SHOWN ON A STATEMENT."
600 PRINT A$
610 INPUT R$
620 IF R$ = "END" THEN 690
630 IF VAL(R$) > 0 THEN 660
640 PRINT E$:GOTO 600
660 NC = NC+1:C(NC) = VAL(R$):TC =TC+C(NC)
```

```
670 IF NC < MC THEN 610
680 PRINT L$
690 PRINT"TOTAL =";TC:PRINT
700 PRINT"NOW ENTER THE AMOUNTS OF ANY"
710 PRINT"SERVICE CHARGES OR DEBIT MEMOS."
720 PRINT A$
730 INPUT R$
740 IF R$ = "END" THEN 790
750 IF VAL(R$) > 0 THEN 770
760 PRINT E$:GOTO 720
770 NS = NS+1:TS = TS+VAL(R$)
780 GOTO 730
790 PRINT"TOTAL =";TS:PRINT
800 GOSUB 2000
810 W = SB+TD+TS-CB-TC-TR:W = ABS(W)
815 IF W > .001 THEN 840
820 W = 0:PRINT"IT BALANCES !!!"
830 GOTO 850
840 PRINT"SORRY, IT'S OUT OF BALANCE."
850 PRINT"STATEMENT BALANCE +"
860 PRINT"DEPOSITS OUTSTANDING +"
870 PRINT"SERVICE CHARGES =";SB+TD+TS
880 PRINT:PRINT"CHECKBOOK BALANCE +"
890 PRINT"CHECKS OUTSTANDING +"
900 PRINT"RECENT DEPOSITS =";CB+TC+TR
910 PRINT:PRINT"DIFFERENCE =";W
920 PRINT
930 END
2000 PRINT"ENTER THE AMOUNT OF ANY DEPOSIT"
2010 PRINT"SHOWN IN YOUR CHECKBOOK AFTER"
2020 PRINT"CHECK NO.";LC;"THAT IS ALSO"
2030 PRINT"LISTED IN THE STATEMENT."
2050 PRINT A$
2060 INPUT R$
2070 IF R$ = "END" THEN 2130
2080 IF VAL(R$) > 0 THEN 2100
2090 PRINT E$:GOTO 2050
2100 NR = NR+1:TR = TR+VAL(R$)
2110 GOTO 2060
2130 PRINT"TOTAL =";TR:PRINT
2140 RETURN

READY.
```

EASY CHANGES

Change the limitation of how many outstanding checks you can enter. Line 150 establishes this limit. If you have more than 10 checks outstanding at some time, change the value of MC to 100, for example.

MAIN ROUTINES

120- 290	Initializes variables and displays first instructions.
300- 340	Gets most recent check number.
380- 410	Gets checkbook balance after most recent check number.
430- 540	Gets outstanding deposits.
550- 690	Gets outstanding checks.
700- 790	Gets service charges and debit memos.
800	Gets recent deposits and credit memos.
810- 930	Does balancing calculation. Displays it. Ends program.
2000-2140	Subroutine to get recent deposits.

MAIN VARIABLES

MC	Maximum number of checks outstanding.
C	Array for checks outstanding.
TC	Total of checks outstanding.
TD	Total of deposits outstanding.
TS	Total of service charges and debit memos.
TR	Total of recent deposits and credit memos.
NC	Number of checks outstanding.
ND	Number of deposits outstanding.
NS	Number of service charges and debit memos.
NR	Number of recent deposits and credit memos.
E$	Error message.
SB	Statement balance.
LC	Number of last check on statement.
CB	Checkbook balance after last check on statement.
R$	Reply from operator.
W	Amount by which checkbook is out of balance.
A$	Message showing how to indicate no more data.
L$	Message indicating no more room for data.

SUGGESTED PROJECTS

1. Add more informative messages and a more complete introduction to make the program a tutorial for someone who has never balanced a checkbook before.
2. Save all entries from the operator and allow any of them to be reviewed and/or modified if found to be incorrect.

3. If the checkbook is out of balance, have the program do an analysis (as suggested in the "How To Use It" section) and suggest the most likely errors that might have caused the condition.
4. Allow the operator to find arithmetic errors in the checkbook. Ask for the starting balance, then ask for each check or deposit amount. Add or subtract, depending on which type the operator indicates. Display the new balance after each entry so the operator can compare with the checkbook entry.

DECIDE

PURPOSE

"Decisions, decisions!" How many times have you uttered this lament when confronted by a difficult choice? Wouldn't a trusty advisor be helpful on such occasions? Well, you now have one—your Commodore 64 Computer of course.

This program can help you make decisions involving the selection of one alternative from several choices. It works by prying relevant information from you and then organizing it in a meaningful, quantitative manner. Your best choice will be indicated and all of the possibilities given a relative rating.

You can use the program for a wide variety of decisions. It can help with things like choosing the best stereo system, saying yes or no to a job or business offer, or selecting the best course of action for the future. Everything is personalized to your individual decision.

HOW TO USE IT

The first thing the program does is ask you to categorize the decision at hand into one of these three categories:

1) Choosing an item (or thing),
2) Choosing a course of action, or
3) Making a yes or no decision.

You simply press 1, 2, or 3 followed by the **RETURN** key to indicate which type of decision is facing you. If you are choosing an item, you will be asked what type of item it is.

If the decision is either of the first two types, you must next enter a list of all the possibilities under consideration. A question mark will prompt you for each one. When the list is complete, type "END" in response to the last question mark. You must, of course, enter at least two possibilities. (We hope you don't have trouble making decisions from only one possibility!) After the list is finished, it will be re-displayed so that you can verify that it is correct. If not, you must re-enter it.

Now you must think of the different factors that are important to you in making your decision. For example, location, cost, and quality of education might govern the decision of which college to attend. For a refrigerator purchase, the factors might be things like price, size, reliability, and warranty. In any case, you will be prompted for your list with a succession of question marks. Each factor is to be entered one at a time with the word "END" used to terminate the list. When complete, the list will be re-displayed. You must now decide which single factor is the most important and input its number. (You can enter 0 if you wish to change the list of factors.)

The program now asks you to rate the importance of each of the other factors relative to the most important one. This is done by first assigning a value of 10 to the main factor. Then you must assign a value from 0-10 to each of the other factors. These numbers reflect your assessment of each factor's relative importance as compared to the main one. A value of 10 means it is just as important; lesser values indicate how much less importance you place on it.

Now you must rate the decision possibilities with respect to each of the importance factors. Each importance factor will be treated separately. Considering *only* that importance factor, you must rate how each decision possibility stacks up. The program first assigns a value of 10 to one of the decision possibilities. Then you must assign a relative number (lower, higher, or equal to 10) to each of the other decision possibilities.

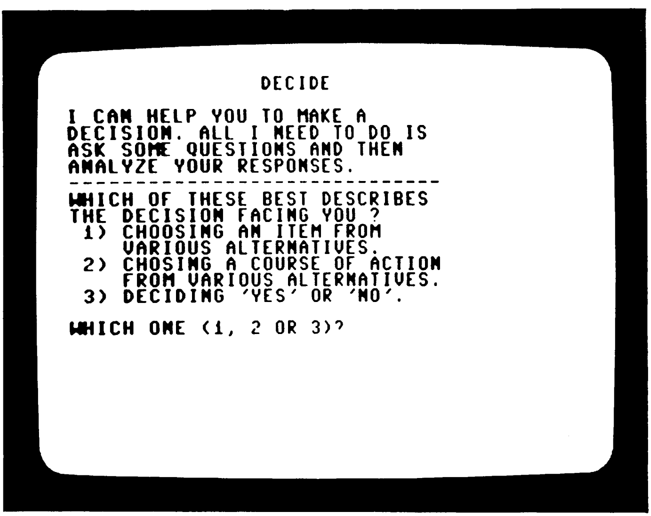
An example might alleviate possible confusion here. Suppose you are trying to decide whether to get a dog, cat, or canary for a pet. Affection is one of your importance factors. The program assigns a value of 10 to the cat. Considering *only* affection, you might assign a value of 20 to the dog and 6.5 to the canary. This means *you* consider a dog twice as affectionate as a cat but a

canary only about two thirds as affectionate as a cat. (No slighting of bird lovers is intended here, of course. Your actual ratings may be entirely different.)

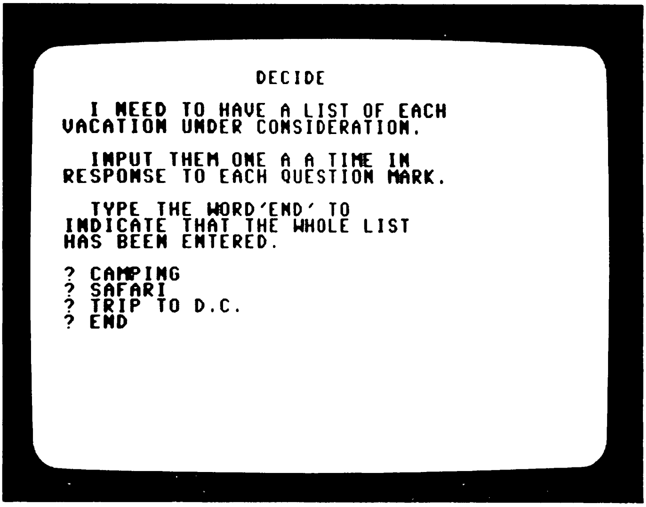
Armed with all this information, the program will now determine which choice seems best for you. The various possibilities are listed in order of ranking. Alongside each one is a relative rating with the best choice being normalized to a value of 100.

Of course, DECIDE should not be used as a substitute for good, clear thinking. However, it can often provide valuable insights. You might find one alternative coming out surprisingly low or high. A trend may become obvious when the program is re-run with improved data. At least, it may help you think about decisions systematically and honestly.

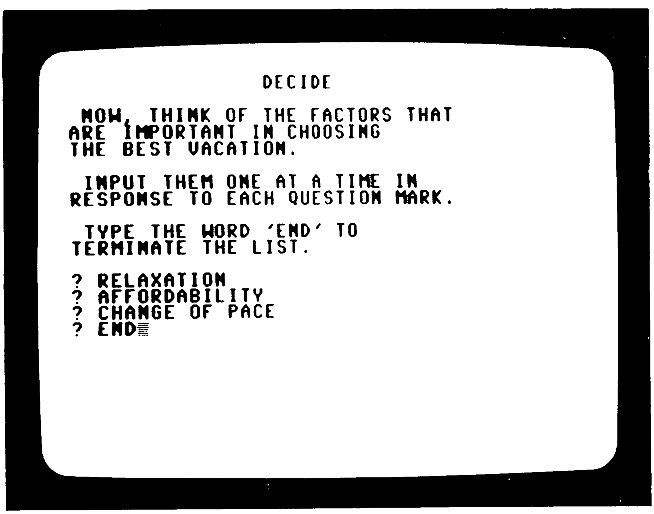
SAMPLE RUN



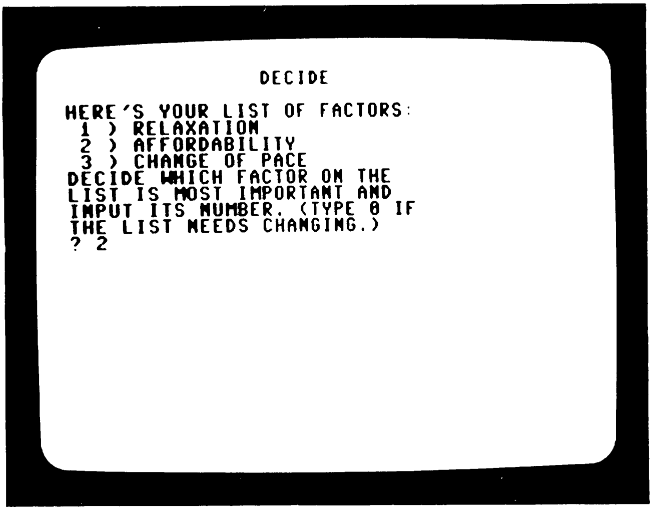
The program displays an introduction, and the operator begins providing the necessary information.



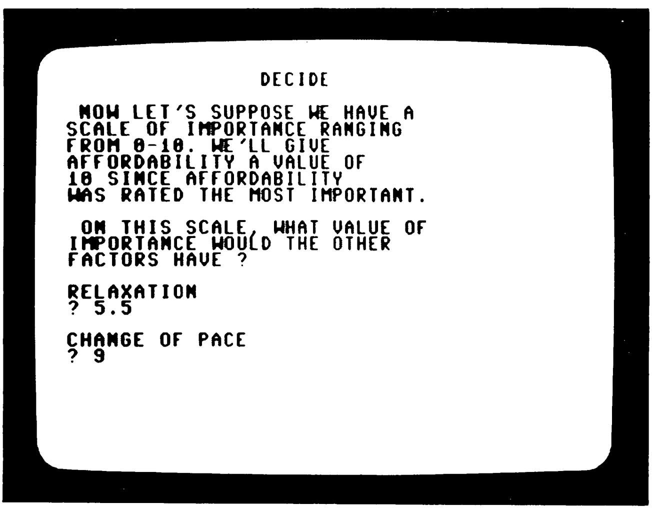
After selecting option 1 and entering a single item "VACATION", the types of vacations are requested.



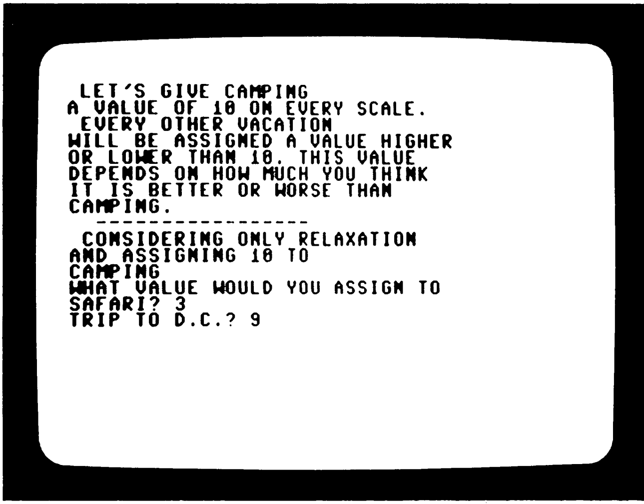
The operator continues by entering the factors that are important in choosing the best vacation.



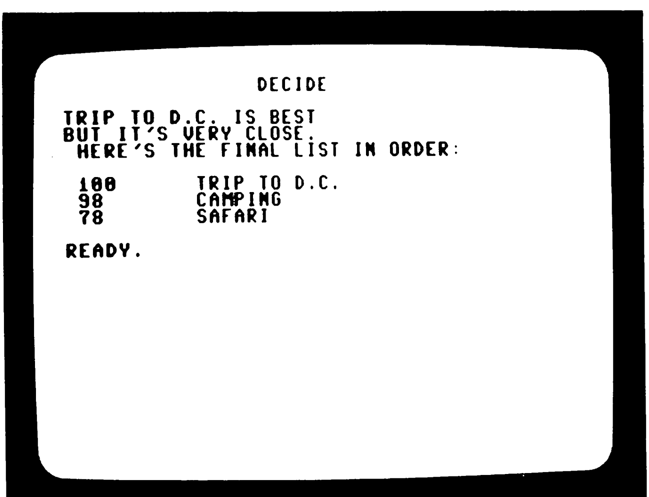
The operator selects the most important factor.



The operator selects a value of importance for the factors.



The operator rates each vacation based on each factor.



The program displays the list of vacations in order.

PROGRAM LISTING

```
READY.

100 REM: DECIDE
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 CLR
160 MD = 10
170 DIM L$(MD), F$(MD), V(MD)
180 DIM C(MD,MD), D(MD), Z(MD)
190 E$ = "END"
200 GOSUB 5000
210 PRINT"I CAN HELP YOU TO MAKE A"
220 PRINT"DECISION. ALL I NEED TO DO IS"
230 PRINT"ASK SOME QUESTIONS AND THEN"
240 PRINT"ANALYZE YOUR RESPONSES."
250 FOR J = 1 TO 30:PRINT"-";:NEXT
260 PRINT
270 PRINT"WHICH OF THESE BEST DESCRIBES"
280 PRINT"THE DECISION FACING YOU ?"
290 PRINT" 1) CHOOSING AN ITEM FROM"
300 PRINT"    VARIOUS ALTERNATIVES."
310 PRINT" 2) CHOSING A COURSE OF ACTION"
320 PRINT"    FROM VARIOUS ALTERNATIVES."
330 PRINT" 3) DECIDING 'YES' OR 'NO'."
340 PRINT
350 INPUT"WHICH ONE (1, 2 OR 3)";T
360 IF T < 1 OR T > 3 THEN 200
400 GOSUB 5000
410 ON T GOTO 420,440,460
420 PRINT"WHAT TYPE OF ITEM IS IT"
430 INPUT T$:GOTO 500
440 T$ ="COURSE OF ACTION"
450 GOTO 500
460 T$ ="'YES' OR 'NO'":NI = 2
470 L$(1) ="DECIDING YES"
480 L$(2) ="DECIDING NO"
490 GOTO 900
500 GOSUB 5000:NI = 0
510 PRINT"  I NEED TO HAVE A LIST OF EACH"
520 PRINT T$;" UNDER CONSIDERATION."
530 PRINT
540 PRINT"  INPUT THEM ONE A A TIME IN"
550 PRINT"RESPONSE TO EACH QUESTION MARK."
560 PRINT
570 PRINT"  TYPE THE WORD'";E$;"' TO"
580 PRINT"INDICATE THAT THE WHOLE LIST"
590 PRINT"HAS BEEN ENTERED.":PRINT
600 IF NI < MD THEN 620
```

```
610 PRINT"-LIST FULL-":GOTO 650
620 NI = NI+1:INPUT L$(NI)
630 IF L$(NI) <> E$ THEN 600
640 NI = NI-1
650 IF NI >=2 THEN 700
660 PRINT
670 PRINT"YOU NEED AT LEAST 2 CHOICES !!"
680 PRINT:PRINT"TRY AGAIN"
690 GOSUB 5200:GOTO 500
700 GOSUB 5000
710 PRINT"OK. HERE'S YOUR LIST:"
720 PRINT:FOR J = 1 TO NI
730 PRINT J;CHR$(8);") ";L$(J)
740 NEXT:PRINT
750 FOR J = 1 TO 9:GET R$:NEXT
760 INPUT"IS THE LIST CORRECT (Y OR N)";R$
770 IF R$ = "Y" THEN 900
780 IF R$ <> "N" THEN 700
790 PRINT
800 PRINT"THE LIST MUST BE RE-ENTERED"
810 GOSUB 5200:GOTO 500
900 GOSUB 5000:GET R$
910 PRINT" NOW, THINK OF THE FACTORS THAT"
920 IF T<3 THEN PRINT"ARE IMPORTANT IN CHOOSING"
930 IF T < 3 THEN PRINT"THE BEST ";T$;"."
940 IF T = 3 THEN PRINT"ARE IMPORTANT TO YOU IN"
950 IF T = 3 THEN PRINT"DECIDING ";T$;"."
960 PRINT:PRINT" INPUT THEM ONE AT A TIME IN"
970 PRINT"RESPONSE TO EACH QUESTION MARK."
980 PRINT:PRINT" TYPE THE WORD '";E$;"' TO"
990 PRINT"TERMINATE THE LIST."
1000 PRINT:NF = 0
1010 IF NF >= MD THEN PRINT"-- LIST FULL --"
1020 IF NF >= MD THEN 1060
1030 NF = NF+1:INPUT F$(NF)
1040 IF F$(NF) <> E$ THEN 1010
1050 NF = NF-1:PRINT
1060 IF NF < 1 THEN PRINT"YOU MUST HAVE AT
     LEAST 1 -"
1065 IF NF < 1 THEN PRINT"PLEASE REDO."
1070 IF NF < 1 THEN GOSUB 5200
1080 IF NF < 1 THEN 900
1100 GOSUB 5000
1110 PRINT"HERE'S YOUR LIST OF FACTORS:"
1130 FOR J = 1 TO NF
1140 PRINT J;CHR$(8);") ";F$(J)
1150 NEXT
1160 PRINT"DECIDE WHICH FACTOR ON THE"
1170 PRINT"LIST IS MOST IMPORTANT AND"
1180 PRINT"INPUT ITS NUMBER. (TYPE 0 IF"
1190 PRINT"THE LIST NEEDS CHANGING.)"
1200 INPUT A:A = INT(A)
```

```
1210 IF A = 0 THEN 900
1220 IF A > NF OR A < 0 THEN 1100
1300 GOSUB 5000
1310 IF NF = 1 THEN 1500
1320 PRINT" NOW LET'S SUPPOSE WE HAVE A"
1330 PRINT"SCALE OF IMPORTANCE RANGING"
1340 PRINT"FROM 0-10. WE'LL GIVE"
1350 PRINT F$(A);" A VALUE OF"
1360 PRINT "10 SINCE ";F$(A)
1370 PRINT"WAS RATED THE MOST IMPORTANT."
1380 PRINT:PRINT" ON THIS SCALE, WHAT VALUE OF"
1390 PRINT"IMPORTANCE WOULD THE OTHER"
1400 PRINT"FACTORS HAVE ?"
1410 FOR J = 1 TO NF
1420 IF J = A THEN 1490
1430 PRINT:PRINT F$(J)
1440 INPUT V(J)
1450 IF V(J) <0 THEN 1480
1460 IF V(J) >10 THEN 1480
1470 GOTO 1490
1480 PRINT" IMPOSSIBLE VALUE - TRY AGAIN"
1485 GOTO 1430
1490 NEXT
1500 V(A) = 10:Q = 0:FOR J = 1 TO NF
1510 Q = Q+V(J):NEXT:FOR J = 1 TO NF
1520 V(J) =V(J)/Q:NEXT:GOSUB 5000
1530 IF T <> 3 THEN PRINT" EACH ";T$;" MUST NOW"
1540 IF T = 3 THEN PRINT" DECIDING 'YES' OR"
1550 IF T = 3 THEN PRINT"DECIDING 'NO' MUST NOW"
1560 PRINT"BE COMPARED WITH RESPECT TO"
1570 PRINT"EACH IMPORTANCE FACTOR."
1580 PRINT" WE'LL CONSIDER EACH FACTOR"
1590 PRINT"SEPARATELY AND THEN RATE"
1600 IF T <> 3 THEN PRINT"EACH ";T$;" IN TERMS"
1610 IF T = 3 THEN PRINT"DECIDING 'YES' OR"
1620 IF T = 3 THEN PRINT"DECIDING 'NO' IN TERMS"
1630 PRINT"OF THAT FACTOR ONLY"
1634 PRINT"*** (HIT ANY KEY TO CONTINUE)"
1638 GET R$:IF R$ = "" THEN 1638
1639 PRINT CHR$(147)
1640 PRINT:PRINT" LET'S GIVE ";L$(1)
1650 PRINT"A VALUE OF 10 ON EVERY SCALE."
1660 IF T <> 3 THEN PRINT" EVERY OTHER ";T$
1670 IF T = 3 THEN PRINT" THEN DECIDING 'NO'"
1680 PRINT"WILL BE ASSIGNED A VALUE HIGHER"
1690 PRINT"OR LOWER THAN 10. THIS VALUE"
1700 PRINT"DEPENDS ON HOW MUCH YOU THINK"
1710 PRINT"IT IS BETTER OR WORSE THAN"
1720 PRINT L$(1);"."
1800 FOR J = 1 TO NF
1810 PRINT"  -----------------------"
1820 PRINT" CONSIDERING ONLY ";F$(J)
```

```
1830 PRINT"AND ASSIGNING 10 TO"
1835 PRINT L$(1)
1840 PRINT"WHAT VALUE WOULD YOU ASSIGN TO"
1850 FOR K = 2 TO NI
1860 PRINT L$(K);:INPUT C(K,J)
1870 IF C(K,J) >= 0 THEN 1900
1880 PRINT" -- NEGATIVE VALUES ILLEGAL -- "
1890 GOTO 1860
1900 NEXT:PRINT:C(1,J) = 10:NEXT
2000 FOR J = 1 TO NF:Q = 0
2010 FOR K = 1 TO NI
2020 Q = Q+C(K,J):NEXT
2030 FOR K = 1 TO NI
2040 C(K,J) = C(K,J)/Q:NEXT:NEXT
2050 FOR K = 1 TO NI:D(K) = 0
2060 FOR J = 1 TO NF
2070 D(K) = D(K)+C(K,J)*V(J):NEXT
2080 NEXT:FOR K = 1 TO NI
2090 IF D(K) > MX THEN MX = D(K)
2100 NEXT: FOR K = 1 TO NI
2110 D(K) = D(K)*100/MX:NEXT
2200 FOR K = 1 TO NI:Z(K) = K:NEXT
2210 NM = NI-1:FOR K = 1 TO NI
2220 FOR J = 1 TO NM:N1 = Z(J)
2230 N2 = Z(J+1)
2240 IF D(N1) > D(N2) THEN 2260
2250 Z(J+1) = N1:Z(J) = N2
2260 NEXT:NEXT:J1 = Z(1):J2 = Z(2)
2270 DF =D(J1)-D(J2):GOSUB 5000
2300 PRINT L$(J1);" IS BEST"
2310 IF DF < 5 THEN PRINT"BUT IT'S VERY CLOSE."
2320 IF DF < 5 THEN 2380
2330 IF DF < 10 THEN PRINT"BUT IT'S FAIRLY CLOSE."
2340 IF DF < 10 THEN 2380
2350 IF DF < 20 THEN PRINT"BY A FAIR AMOUNT."
2360 IF DF < 20 THEN 2380
2370 PRINT"QUITE DECISIVELY."
2380 PRINT" HERE'S THE FINAL LIST IN ORDER:"
2390 PRINT
2480 FOR J = 1 TO NI:Q = Z(J)
2490 PRINT INT(D(Q)),L$(Q):NEXT
3000 END
5000 FOR J = 1 TO 500:NEXT
5010 PRINT CHR$(147):PRINT TAB(15);"DECIDE"
5020 PRINT:RETURN
5200 FOR J = 1 TO 1500:NEXT:RETURN

READY.
```

EASY CHANGES

1. The word "END" is used to flag the termination of various input lists. If you wish to use something else (because of

conflicts with items on the list), change the definition of E$ in line 190. For example, to use the word "DONE," change line 190 to

```
190 E$="DONE"
```

2. Line 5200 contains a timing delay used regularly in the program. If things seem to change too fast, you can make the number 1500 larger. Try

```
5200 FOR J=1 TO 3000:NEXT: RETURN
```

3. The program can currently accept up to ten decision alternatives and/or ten importance factors. If you need more, increase the value of MD in line 160. Thus, to use 15 values, line 160 should be

```
160 MD=15
```

MAIN ROUTINES

150- 190	Initializes and dimensions variables.
200- 360	Determines category of decision.
400- 490	Gets or sets T$.
500- 810	Gets list of possible alternatives from user.
900-1220	Gets list of importance factors from user.
1300-1490	User rates each importance factor.
1500-1900	User rates the decision alternatives with respect to each importance factor.
2000-2110	Evaluates the various alternatives.
2200-2270	Sorts alternatives into their relative ranking.
2300-3000	Displays results.
5000-5020	Subroutine to clear screen and display header.
5200	Time wasting subroutine.

MAIN VARIABLES

MD	Maximum number of decision alternatives.
NI	Number of decision alternatives.
NM	NI-1.
L$	String array of the decision alternatives.
NF	Number of importance factors.
F$	String array of the importance factors.
V	Array of the relative values of each importance factor.
A	Index number of most important factor.

C	Array of relative values of each alternative with respect to each importance factor.
T	Decision category (1 = item, 2 = course of action, 3 = yes or no).
T$	String name of decision category.
E$	String to signal the end of an input data list.
J,K	Loop indices.
R$	User reply string.
Q,N1,N2	Work variables.
D	Array of each alternative's value.
MX	Maximum value of all alternatives.
DF	Rating difference between best two alternatives.
Z	Array of the relative rankings of each alternative.

SUGGESTED PROJECTS

1. Allow the user to review his numerical input and modify it if desired.
2. Insights into a decision can often be gained by a sensitivity analysis. This involves running the program a number of times for the same decision. Each time, one input value is changed (usually the one you are least confident about). By seeing how the results change, you can determine which factors are the most important. Currently, this requires a complete rerunning of the program each time. Modify the program to allow a change of input after the regular output is produced. Then recalculate the results based on the new values. (Note that many input arrays are clobbered once all the input is given. This modification will require saving the original input in new arrays so that it can be reviewed later.)

LOAN

PURPOSE

One of the most frustrating things about borrowing money from a bank (or credit union or Savings and Loan) is that it's not easy to fully evaluate your options. When you are borrowing from a credit union to buy a new car, you might have the choice of a thirty-six or a forty-eight month repayment period. When buying a house, you can sometimes get a slightly lower interest rate for your loan if you can come up with a larger down payment. Which option is best for you? How will the monthly payment be affected? Will there be much difference in how fast the principal of the loan decreases? How much of each payment will be for interest, which is tax-deductible?

You need to know the answers to all these questions to make the best decision. This program gives you the information you need.

HOW TO USE IT

The program first asks you the size of the loan you are considering. Only whole dollar amounts are allowed—no pennies. Loans of one million dollars or more are rejected (you can afford to hire an investment counselor if you want to borrow that much). Then you are asked the yearly interest rate for the loan. Enter this number as a percentage, such as "10.8." Next, you are asked to give the period of the loan in months. For a five year loan, enter 60. For a thirty year mortgage, enter 360. The program then displays this information for you and calculates the

monthly payment that will cause the loan to be paid off with equal payments each month over the life of the loan.

At this point you have four options. First, you can show a monthly analysis. This displays a month-by-month breakdown, showing the state of the loan after each payment. The four columns of data shown for each month are the payment number (or month number) of the loan, the remaining balance of the loan after that payment, the amount of that payment that was interest, and the accumulated interest paid to date. Twelve lines of data are displayed on the screen, and then you can either press the **T** key to get the final totals for the loan, or any other key to get the data for the next twelve months of the loan.

The second option is overriding the monthly payment. It is a common practice with second mortgage loans to make smaller monthly payments each month with a large "balloon" payment as the final payment. You can use this second option to try various monthly payments to see how they affect that big payment at the end. After overriding the monthly payment, you will want to use the first option next to get a monthly analysis and final totals using the new monthly payment.

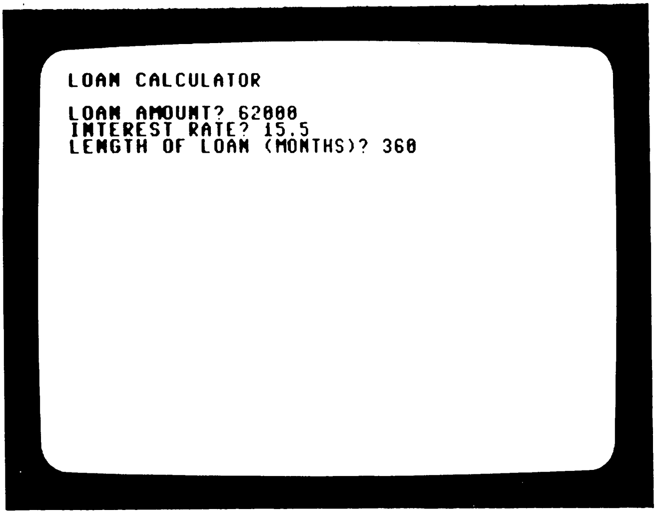
The third option is to simply start over. You will generally use this option if you are just comparing what the different monthly payments would be for different loan possibilities.

The fourth option ends the program.

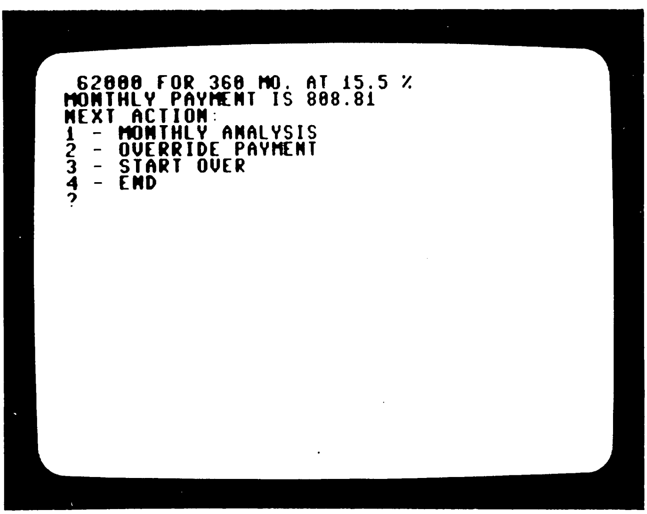
By the way, there is a chance that the monthly payment calculated by your lender will differ from the one calculated here by a penny or two. We like to think that this is because we are making a more accurate calculation.

NOTE: SEE DISCLAIMER IN FRONT PART OF BOOK

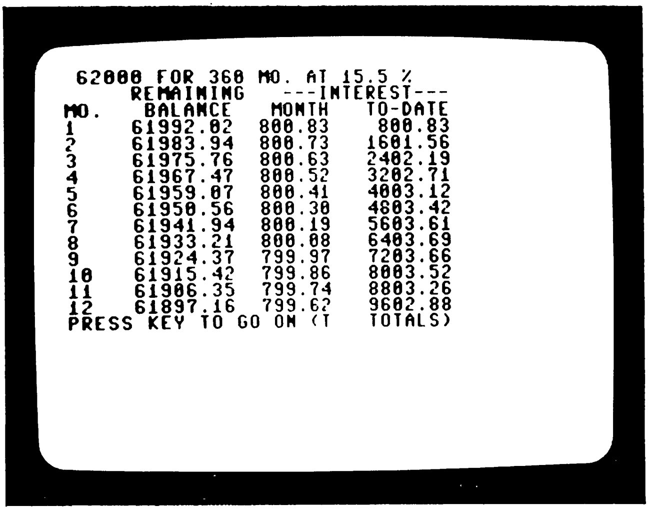
SAMPLE RUN



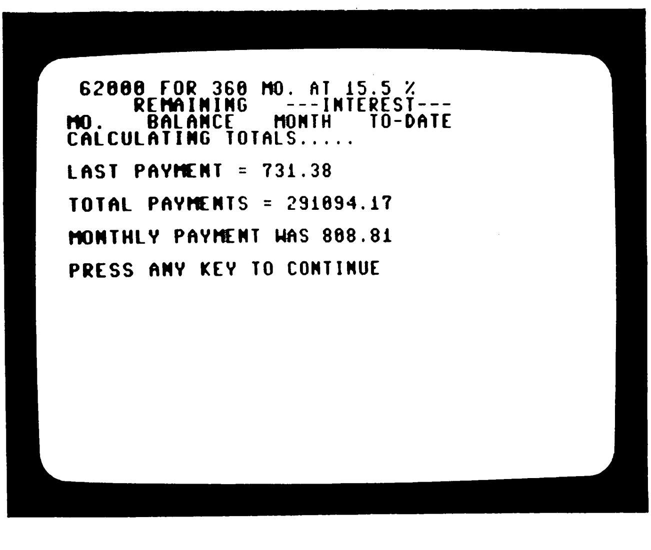
The operator enters the three necessary pieces of information about his or her loan.



The program responds with the monthly payment that will pay off the loan with equal payments over its life, then asks the operator what to do next. The operator will request a monthly analysis.



The program responds with information about the first twelve months of the loan, then waits.



The operator presses “T”, and after a few seconds the program displays totalling information about the loan.

PROGRAM LISTING

```
READY.

100 REM: LOAN CALCULATOR
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 CLR:PRINT CHR$(147):BL$="          "
130 PRINT"LOAN CALCULATOR"
140 PRINT
150 INPUT"LOAN AMOUNT";A
155 GOSUB 1000:IF A = 0 THEN 150
160 INPUT"INTEREST RATE";R
170 INPUT"LENGTH OF LOAN (MONTHS)";N
180 R = ABS(R):M = R/1200
190 GOSUB 800:W = 1
200 FOR J = 1 TO N:W = W*(1+M):NEXT
210 P = (A*M*W)/(W-1)
220 P = INT(P*100+.99):P = P/100
230 PRINT"MONTHLY PAYMENT IS";P
240 FP = P
250 PRINT"NEXT ACTION:"
270 PRINT"1 - MONTHLY ANALYSIS"
280 PRINT"2 - OVERRIDE PAYMENT"
290 PRINT"3 - START OVER"
300 PRINT"4 - END"
310 INPUT C
320 ON C GOTO 440,400,120,370
330 PRINT"CHOICES ARE 1,2,3,4"
340 GOTO 250
370 END
400 PRINT
410 INPUT"MONTHLY PAYMENT";P
420 GOTO 240
440 GOSUB 450:GOTO 510
450 GOSUB 800
460 PRINT TAB(5);"REMAINING";
470 PRINT TAB(17);"---INTEREST---"
480 PRINT"MO.    BALANCE";TAB(16);
490 PRINT"MONTH    TO-DATE"
500 RETURN
510 B = A*100:TT = 0:TP = 0:L = 0
520 P = P*100:R$ = "":FOR J = 1 TO N
530 T = M*B:T = INT(T+.5)
540 IF J = N THEN P = B+T
550 TP = TP+P:B = B-P+T:TT = TT+T
560 IF B < 0 THEN GOSUB 2000
570 IF R$ = "T" THEN 690
580 W = B:GOSUB 900:B$ = S$
590 W = T:GOSUB 900:T$ = RIGHT$(S$,8)
600 W = TT:GOSUB 900:TT$ = " "+S$
```

```
605 J$ = STR$(J):J$ = RIGHT$(J$,LEN(J$)-1)
610 PRINT J$;TAB(4);B$;T$;TT$
620 L = L+1:IF L < 12 THEN 690
630 PRINT"PRESS KEY TO GO ON";
640 PRINT" (T = TOTALS)";
650 GET R$:IF R$ = "" THEN 650
660 L = 0:GOSUB 450
670 IF R$ <> "T" THEN 690
680 PRINT"CALCULATING TOTALS....."
690 NEXT
700 PRINT:PRINT"LAST PAYMENT =";
710 PRINT P/100
720 PRINT:PRINT"TOTAL PAYMENTS =";
730 PRINT TP/100
740 PRINT:PRINT"MONTHLY PAYMENT WAS";FP
750 PRINT
760 PRINT"PRESS ANY KEY TO CONTINUE"
770 GET X$:IF X$ = "" THEN 770
780 P = FP:GOTO 240
800 PRINT CHR$(147):PRINT A;"FOR";N;
810 PRINT"MO. AT";R;"%"
820 RETURN
900 W = INT(W):S$ = STR$(W)
910 K = LEN(S$)-1:S$ = MID$(S$,2,K)
920 IF K = 1 THEN S$ = BL$+".0"+S$:RETURN
930 IF K = 2 THEN S$ = BL$+"."+S$:RETURN
940 D$ = "."+RIGHT$(S$,2)
950 S$ = LEFT$(S$,K-2)+D$
960 S$ = LEFT$(BL$,8-K)+S$
970 RETURN
1000 A = ABS(A):A = INT(A)
1010 IF A < 1000000 THEN RETURN
1020 PRINT"TOO LARGE AN AMOUNT"
1030 A = 0:RETURN
2000 P = P+B:TP = TP+B:B = 0:RETURN

READY.
```

EASY CHANGES

1. The number of lines of data displayed on each screen when getting a monthly analysis can be changed by altering the constant 12 in statement 620.
2. To include the monthly payment in the heading at the top of each screen of the monthly analysis, insert the following line:

```
815 IF FP<>0 THEN PRINT"MONTHLY PAYMENT IS";FP
```

MAIN ROUTINES

120- 170	Displays title. Gets loan information.
200- 230	Calculates and displays monthly payment.
250- 370	Asks for next action. Goes to corresponding routine.
400- 410	Gets override for monthly payment.
440- 780	Calculates and displays monthly analysis.
800- 820	Subroutine to clear screen and display data about the loan at the top.
900- 970	Subroutine to convert integer amount to fixed-length string with aligned decimal point.
1000-1030	Edits loan amount (size and whole dollar).
2000	Subroutine to handle early payoff of loan.

MAIN VARIABLES

BL$	String of 6 blank spaces.
A	Amount of loan.
R	Interest rate (percentage).
N	Length of loan (number of months).
M	Monthly interest rate (not percentage).
W	Work variable.
P	Monthly payment (times 100).
FP	First monthly payment.
C	Choice of next action.
B	Remaining balance of loan (times 100).
TT	Total interest to date (times 100).
TP	Total payments to date.
L	Number of lines of data on screen.
R$	Reply from operator at keyboard.
J	Work variable for loops.
T	Monthly interest.
B$	Remaining balance to be displayed (two decimal places).
T$	Monthly interest to be displayed (two decimal places).
TT$	Total interest to be displayed (two decimal places).
S$,D$	Work strings.
K	Work variable.

SUGGESTED PROJECTS

1. Display a more comprehensive analysis of the loan along with the final totals. Show the ratio of total payments to the amount of the loan (TP divided by A), for example.
2. Modify the program to show an analysis of resulting monthly payments for a range of interest rates and/or loan lengths near those provided by the operator. For example, if an interest rate of 9.5 percent was entered, display the monthly payments for 8.5, 9, 9.5, 10, and 10.5 percent.

MILEAGE

PURPOSE

For many of us, automobile operating efficiency is a continual concern. This program can help by keeping track of gasoline consumption, miles driven, and fuel mileage for a motor vehicle. It allows reading and writing data files with the cassette unit. Thus, a master data file may be retained and updated. The program computes mileage (miles per gallon or MPG) obtained after each gasoline fill-up. A running log of all information is maintained. This enables trends in vehicle operation efficiency to be easily checked.

HOW TO USE IT

The program requests the following data from the operator as a record of each gasoline fill-up: date, odometer reading, and number of gallons purchased. The most useful results will be obtained if entries are chronological and complete, with each entry representing a full gasoline fill-up.

In order to use the cassette features, the operator must be able to position the tape correctly for both reading and writing. The simplest way to do this is to only record files at the beginning of a tape. One tape could certainly be used this way, with each file writing over the previous one. However, we suggest alternating between two physical tapes. This will insure a reasonably up-to-date back-up tape in case of any failure.

The program operates from a central command mode. The operator requests branching to any one of five available subrou-

tines. When a subroutine completes execution, control returns to the command mode for any additional requests. A brief description of each subroutine now follows:

1) READ OLD MASTER FILE

This reads previously stored data from the cassette. Any data already in memory is deleted. During the read, the name of the data file and the total number of records read are displayed.

2) INPUT FROM TERMINAL

This allows data records to be entered directly from the terminal. This mode is used to provide additional information after a cassette read and to enter data for the first time. The program will prompt the operator for the required information and then let him verify that it was entered correctly. A response of "D" to the verification request signals that no more data is to be entered.

3) WRITE NEW MASTER FILE

This command causes the current data to be written on cassette. The program requests a name for the file. When later read, this name will be displayed, allowing verification of the correct data file.

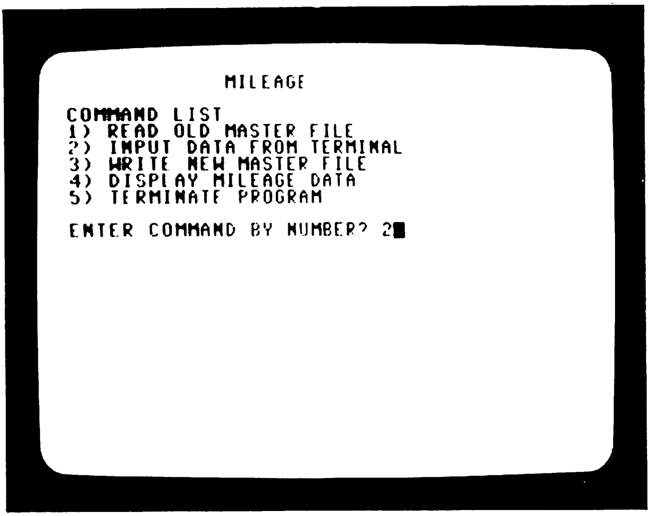
4) DISPLAY MILEAGE DATA

This subroutine computes mileage (miles per gallon) from the available data. It formats all information and displays it in tabular form. Numerical values are rounded to the nearest tenth. When data fills the screen, the user is prompted to hit any key to continue the listing. When all data is displayed, hitting any key will re-enter command mode.

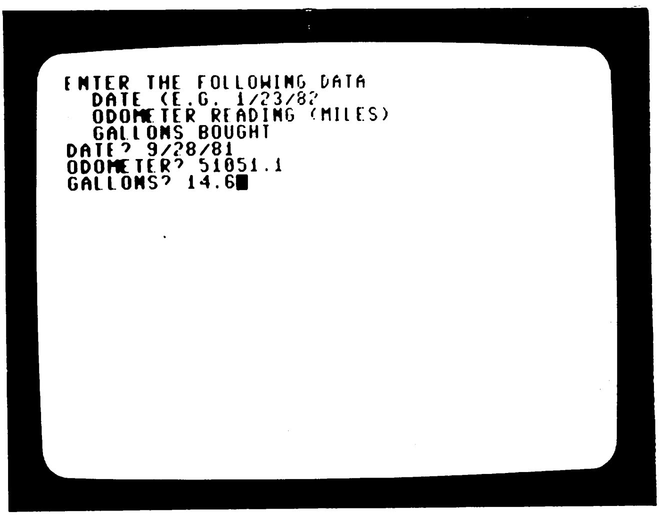
5) TERMINATE PROGRAM

Ends execution and returns the computer to BASIC.

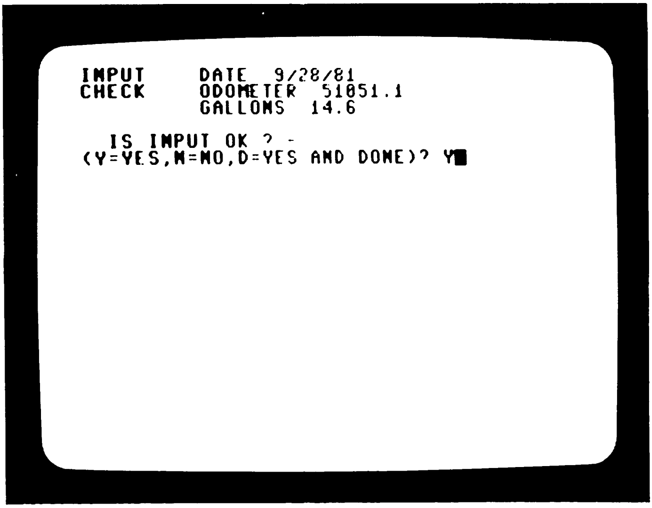
SAMPLE RUN



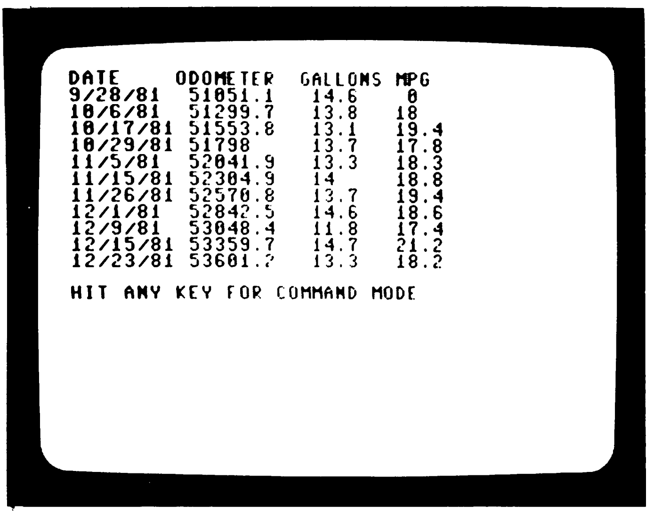
The program's menu is displayed and the operator chooses option 2. This allows data to be entered directly from the terminal.



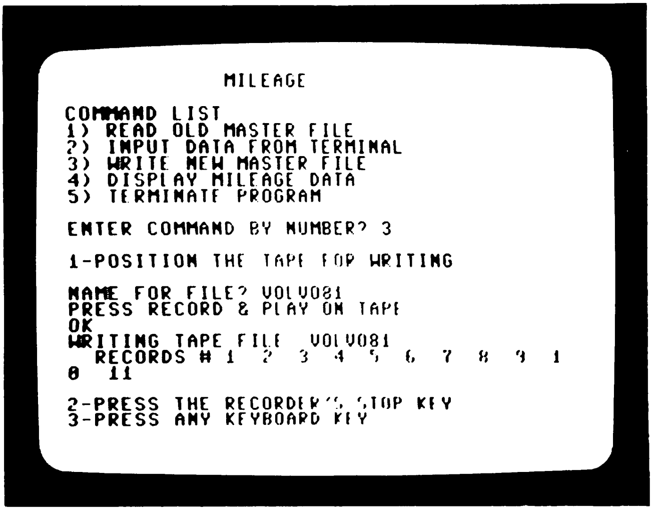
The operator inputs the first data record.



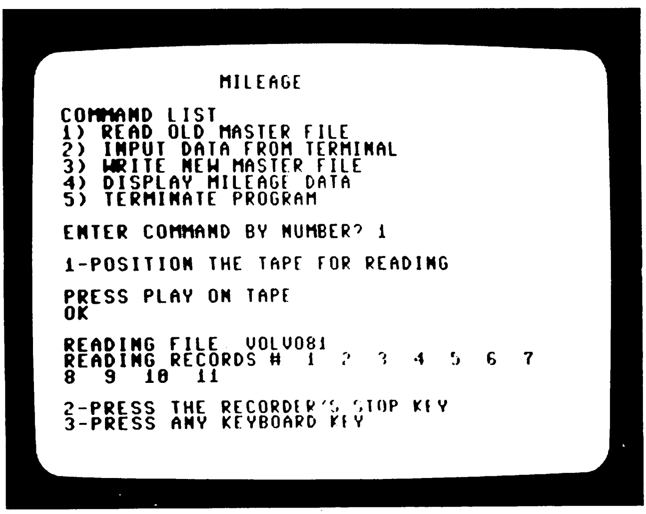
The program then displays the input and the operator confirms that it is correct.



After ten more data records are input, the operator selects option 4 from the command menu. This formats and displays the data along with the fuel MPG obtained. The program will re-enter command mode when a key is hit.



The operator selects option 3 and writes the data to a file called VOLVO81 on cassette tape.



By selecting option 1 in a subsequent run, the file VOLVO81 is retrieved from cassette tape to begin a new session.

PROGRAM LISTING

```
READY.


100 REM: MILEAGE
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
140 CLR
150 MW = 15
160 MR = 20
170 N = 0
180 DIM D$(MR), D(MR), G(MR), M(MR)
200 PRINT CHR$(147):PRINT TAB(12);"MILEAGE"
210 PRINT:R = 0
220 PRINT"COMMAND LIST"
230 PRINT"1) READ OLD MASTER FILE"
240 PRINT"2) INPUT DATA FROM TERMINAL"
250 PRINT"3) WRITE NEW MASTER FILE"
260 PRINT"4) DISPLAY MILEAGE DATA"
270 PRINT"5) TERMINATE PROGRAM"
280 PRINT
290 INPUT"ENTER COMMAND BY NUMBER";R
300 R = INT(R)
310 IF R = 5 THEN 1800
320 ON R GOTO 1500,400,800,1100
330 GOTO 200
400 IF N < MR THEN 440
410 PRINT
420 PRINT"** NO MORE DATA ALLOWED **"
430 GOSUB 2000:GOTO 200
440 PRINT CHR$(147)
450 PRINT"ENTER THE FOLLOWING DATA:"
460 PRINT"- DATE (E.G. 1/23/82"
470 PRINT"- ODOMETER READING (MILES)"
480 PRINT"- GALLONS BOUGHT"
490 N = N+1:INPUT"DATE";R$
500 R$ = LEFT$(R$,8):D$(N) = R$
510 INPUT"ODOMETER";R:D(N) = R
520 IF R < 0 THEN 510
530 INPUT"GALLONS";R:G(N) = R
540 IF R < 0 THEN 530
550 PRINT CHR$(147)
560 PRINT" INPUT    DATE:  ";D$(N)
570 PRINT" CHECK    ODOMETER:";D(N)
580 PRINT TAB(10);"GALLONS:";G(N)
590 PRINT:PRINT" - IS INPUT OK ? - "
600 INPUT" (Y=YES,N=NO,D=YES AND DONE)";R$
610 R$ = LEFT$(R$,1)
620 IF R$ <> "N" THEN 660
```

```
630 N = N-1:PRINT
640 PRINT"PLEASE RE-ENTER LAST DATA"
650 GOTO 490
660 IF R$ = "D" THEN 200
670 IF R$ <> "Y" THEN 590
680 IF N = MR THEN 410
690 GOTO 490
800 IF N > 0 THEN 830
810 PRINT:PRINT"** NO DATA TO WRITE **"
820 GOSUB 2000:GOTO 200
830 R$ = "WRITING":GOSUB 3000
860 PRINT
870 INPUT"NAME FOR FILE";T$
880 K = N:IF N > MW THEN K = MW
890 OPEN 1,1,1,"MILEAGE"
900 PRINT#1,T$:PRINT#1,K
910 K = 1:L = N:IF N <= MW THEN 950
920 PRINT" - ONLY THE LAST";MW;"VALUES"
930 PRINT"   WILL BE WRITTEN"
940 K = N-MW+1
950 PRINT"WRITING TAPE FILE: ";T$
960 PRINT"  RECORDS #";
970 FOR J = K TO L
980 PRINT#1,D$(J)
982 PRINT#1,D(J)
984 PRINT#1,G(J)
990 PRINT J;:NEXT:PRINT
1000 CLOSE 1:P$ = "2":R$ = "3"
1010 PRINT:GOSUB 3200:GOTO 200
1100 IF N > 0 THEN 1130
1110 PRINT:PRINT"** NOT ENOUGH DATA **"
1120 GOSUB 2000:GOTO 200
1130 M(1) = 0:FOR J = 2 TO N
1150 IF G(J) > 0 THEN 1170
1160 M(J) = 0:GOTO 1190
1170 R = (D(J)-D(J-1))/G(J)
1180 M(J) = R:IF R < 0 THEN M(J) = 0
1190 NEXT:K = -11:L = 0
1200 K = K+12:L = L +12
1210 IF L > N THEN L = N
1220 PRINT CHR$(147)
1230 PRINT"DATE     ODOMETER";
1240 PRINT TAB(18);"GALLONS MPG"
1250 FOR J = K TO L:Q = D(J)
1260 GOSUB 4000
1270 IF Q > 99999 THEN Q = 99999
1280 GOSUB 4100
1290 PRINT D$(J);TAB(13-T);Q;
1300 Q =G(J):GOSUB 4000
1310 IF Q > 999 THEN Q = 999
```

```
1320 GOSUB 4100
1330 PRINT TAB(20-T);Q;
1340 Q = M(J):GOSUB 4000
1350 IF Q > 999 THEN Q = 999
1360 GOSUB 4100
1370 PRINT TAB(27-T);Q:NEXT
1380 PRINT:IF L < N THEN 1410
1390 PRINT"HIT ANY KEY FOR COMMAND MODE"
1400 GOSUB 3220:GOTO 200
1410 PRINT"HIT ANY KEY TO CONTINUE"
1420 GOSUB 3220:GOTO 1200
1500 R$ = "READING":GOSUB 3000
1560 OPEN 1,1,0,"MILEAGE"
1570 PRINT:INPUT#1,T$
1580 PRINT"READING FILE: ";T$
1590 INPUT#1,N
1600 IF N <= MR THEN 1640
1610 PRINT
1620 PRINT"** TOO MANY RECORDS ON TAPE **"
1630 GOSUB 2000:END
1640 PRINT"READING RECORDS # ";
1650 FOR J = 1 TO N
1660 INPUT#1,D$(J)
1662 INPUT#1,D(J)
1664 INPUT#1,G(J)
1670 PRINT J;:NEXT:CLOSE 1
1680 PRINT:PRINT:P$ = "2":R$ = "3"
1690 GOSUB 3200:GOTO 200
1800 END
2000 FOR Q = 1 TO 2000:NEXT
2500 RETURN
3000 PRINT
3010 PRINT"1-POSITION THE TAPE FOR ";R$
3020 RETURN
3200 PRINT P$;"-PRESS THE RECORDER'S STOP KEY"
3210 PRINT R$;"-PRESS ANY KEYBOARD KEY"
3220 GET R$
3230 IF R$ = "" THEN 3220
3240 RETURN
4000 Q = Q*10+0.5:Q = INT(Q)/10
4010 RETURN
4100 IF Q > 9999 THEN T = 5:RETURN
4110 IF Q > 999 THEN T = 4:RETURN
4120 IF Q > 99 THEN T = 3:RETURN
4130 IF Q > 9 THEN T = 2:RETURN
4140 T = 1:RETURN

READY.
```

EASY CHANGES

1. Changing the value of MR in line 160 alters the maximum number of data records that the program allows. You may

need to make MR larger to accommodate additional data. To adjust MR, simply change its value in line 160 from its current value of 20 to whatever you choose.

2. Currently, the program will write a maximum of fifteen data records during the cassette write operation. This number can be altered by changing the value of MW in line 150 from its value of fifteen to whatever you choose. Only the most recent MW records will be written to tape if MW is less than the number of available records when a cassette write is issued. If the number of available records is less than MW, then all the records will be written. The value of MW should not be larger than the value of MR.
3. If you do not care about seeing the dates, they can be removed easily. This saves a little typing on data entry. To remove this feature, delete line 460 entirely and change line 490 to read

```
490 N=N+1:PRINT:R$="-----"
```

MAIN ROUTINES

140- 180	Dimensioning and variable initialization.
200- 330	Command mode. Displays available subroutines.
400- 690	Accepts terminal input.
800-1010	Writes data to the cassette unit.
1100-1420	Calculates mileage and displays all information.
1500-1690	Reads data from the cassette unit.
1800	Terminates execution.
2000-2500	Delay loop.
3000-3240	Displays messages for cassette operation.
4000-4010	Numerical rounding subroutine.
4100-4140	Sets TAB arguments for printing.

MAIN VARIABLES

MW	Maximum number of data records to write.
MR	Maximum number of data records in memory.
N	Current number of data records in memory.
D$	Array of dates.
D	Array of odometer readings.
G	Array of gallons per fill-up.
M	Array of mileage per fill-up.
R	Command mode input.
P$,R$	Temporary string variables.

T$	Data file name used in reading or writing with cassette.
J	Work variable, loop index.
K,L	Loop bounds.
Q	Work variable.
T	TAB argument decrement.

SUGGESTED PROJECTS

1. Calculate and print the average MPG over the whole data file. The total miles driven is D(N) – D(1). The total gallons used is the sum of G(J) for J = 2 to N. This calculation can be done at the end of the DISPLAY MILEAGE subroutine. Programming should be done between lines 1370 and 1380.
2. Allow the user the option to write to cassette only the entries since a certain date. Ask which date and search the D$ array for it. Then set MW to the appropriate number of records to write. These changes are to be made at and after line 800 at the beginning of the subroutine to write on cassette.
3. Add a new command option to verify a data file just written to cassette. It would read the tape and compare it to the data already in memory.
4. Add an option to do statistical calculations over a given subset of the data. The operator inputs a beginning and ending date. He is then shown things like average MPG, total miles driven, total gallons purchased, etc.; all computed only over the range requested.
5. Write a subroutine to graphically display MPG. A bar graph might work well.
6. Add a new parameter in each data record—the cost of each fill-up. Then compute things like the total cost of gasoline, miles/dollar, etc.

QUEST/EXAM

PURPOSE

If you've ever had to analyze the results of a questionnaire, or grade a multiple-choice examination, you know what a tedious and time-consuming process it can be. This is particularly true if you need to accumulate statistics for each question showing how many people responded with each possible answer.

With this program, you provide the data, and the computer does the work.

HOW TO USE IT

As currently set up, the program assumes that the questionnaire or exam has 15 questions, that there are four choices per question, and that there are no more than 20 entries (exam papers). If you wish, these limits can be increased. See the Easy Changes section for details.

To start off, the program asks you for the answer key. If you are scoring an exam, provide the correct answers. The program displays "guide numbers" to help you keep track of which answers you are providing. If you are analyzing a questionnaire, you have no answer key, so just press the **RETURN** key.

Now the program asks you to begin providing the answers for each entry. Again, guide numbers are displayed above the area where you are to enter the data so you can more easily provide the proper answer for the proper question number. If no answer was given for a particular question, leave a blank space. However, if the first question was left blank, you will have to enclose

the entire string of answers within quotation marks. This will cause a small problem in keeping your alignment straight with the guide numbers, but you'll get used to it.

If you make a mistake when entering the data, the program will tell you and ask you to re-enter it. This is most commonly caused by either failing to enter the correct number of answers or entering an invalid character instead of an acceptable answer number. Remember that each answer must be either a blank or a number from one to the number of choices allowed per question.

By the way, you can avoid entering blanks for unanswered questions. Suppose you have a maximum of 5 possible answers per question. Simply tell the program there are 6 choices per question. Then, when a question is unanswered, you can enter a 6 instead of leaving it blank.

If you provided an answer key, the program displays the number and percentage correct after each entry before going on to ask for the next one. When you have no more entries, press the **RETURN** key instead of entering a string of answers.

At this point, the program displays four options from which you choose your next action. Here are brief explanations. You can experiment to verify how they work.

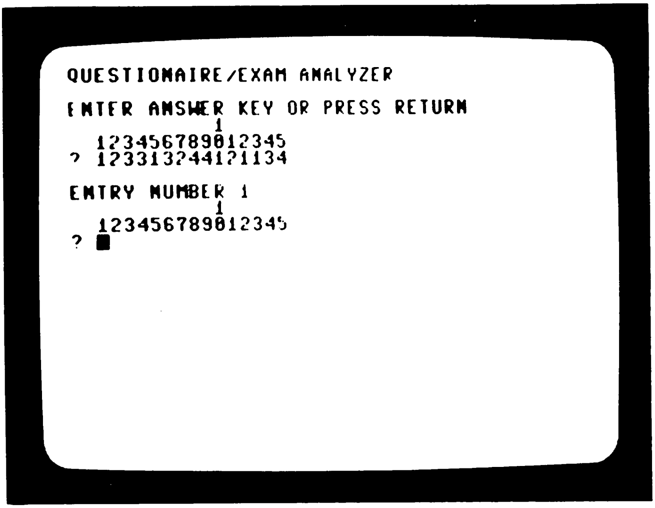
Option one lets you analyze each question, to see how many people responded with each answer. The percentage of people who responded with each answer is also shown. In the case of an exam, the correct answer is indicated with the letter "C" to the right.

Option two allows you to go back and provide more entries. This allows you to pause after entering part of the data, do some analysis of what you have entered so far, and then go back and continue entering data.

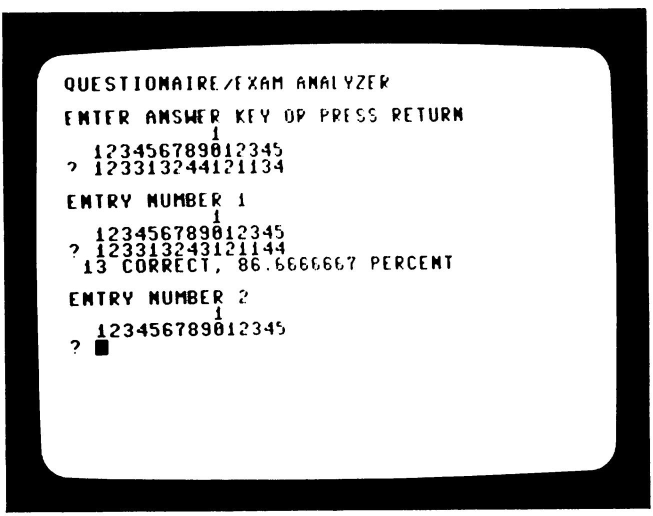
Option three lets you review what you have entered, including the answer key. This permits you to check for duplicate, omitted, or erroneous entries.

Option four ends the program.

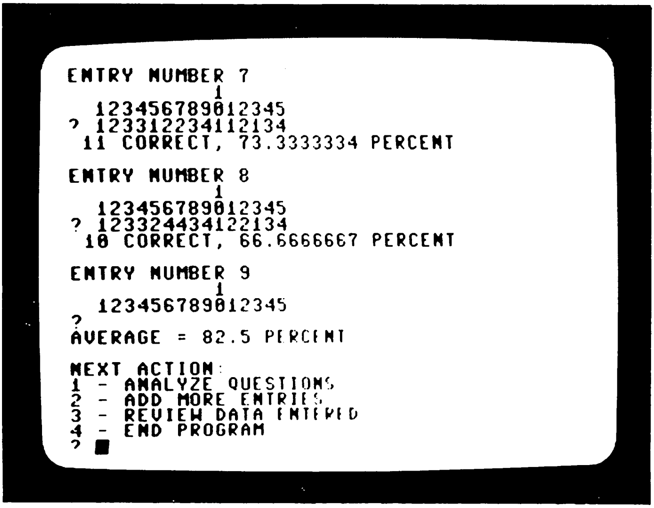
SAMPLE RUN



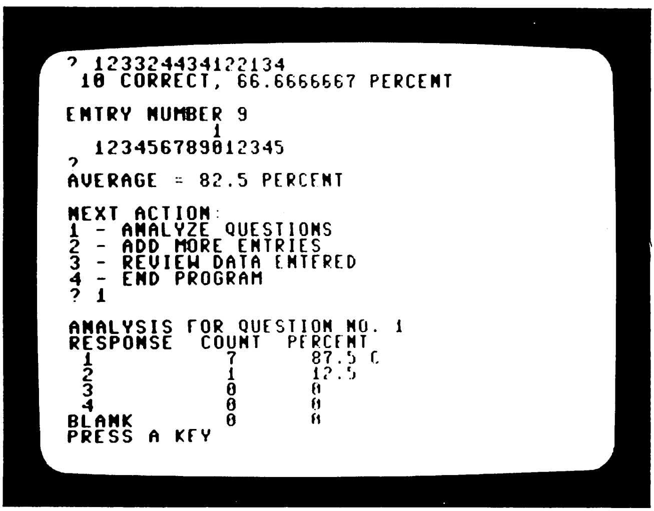
The operator provides the answer key for the examination being scored. The program waits for the data from the first examination paper.



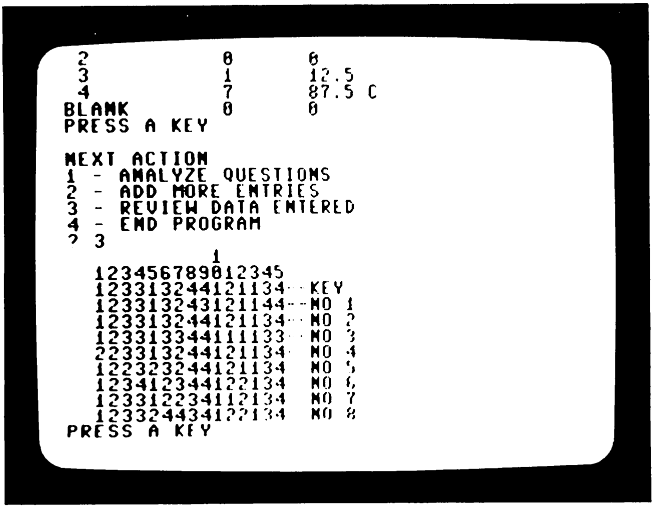
The answers are entered for the first student. The program responds with the number and percentage correct.



Later, instead of providing data for a ninth student, the operator presses the RETURN key, indicating that there are no more entries. The program displays the overall percentage correct, and displays a menu of choices. The operator will pick option 1.



The program provides an analysis of the responses for question number one, then waits for a key to be pressed. Note that seven students answered with number 1, the correct answer.



Later, the operator asks for option 3, which lists the data entered for each of the students.

PROGRAM LISTING

```
READY.

100 REM: QUESTIONAIRE/EXAM
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 CLR:PRINT CHR$(147)
130 PRINT"QUESTIONAIRE/EXAM ANALYZER"
140 E$ = "ERROR. RE-ENTER.":P$ = "PRESS A KEY"
150 Q = 15:C = 4:N = 20
160 DIM Q$(N),C(C)
250 PRINT
260 PRINT"ENTER ANSWER KEY OR PRESS RETURN"
270 GOSUB 900:C$ = RIGHT$(STR$(C),1)
310 INPUT A$:IF LEN(A$) = 0 THEN 340
320 IF LEN(A$) <> Q THEN PRINT E$:GOTO 250
330 T$ = A$:GOSUB 850
335 IF T$ = "B" THEN PRINT E$:GOTO 250
340 K = 1
350 R = 0:PRINT:PRINT"ENTRY NUMBER";K
360 GOSUB 900
370 INPUT Q$(K):W = LEN(Q$(K))
```

```
380 IF W = 0 THEN 500
390 IF W <> Q THEN PRINT E$:GOTO 350
400 T$ = Q$(K):GOSUB 850
405 IF T$ = "B" THEN PRINT E$:GOTO 350
430 IF LEN(A$) = 0 THEN 480
440 FOR J = 1 TO Q
450 IF MID$(A$,J,1) = MID$(Q$(K),J,1) THEN R = R+1
460 NEXT
470 TR = TR+R:PRINT R;"CORRECT,";
475 PRINT R*100/Q;"PERCENT"
480 K = K+1:IF K <= N THEN 350
500 K = K-1:IF LEN(A$) = 0 THEN 520
510 PRINT"AVERAGE =";TR*100/(Q*K);"PERCENT"
520 GOTO 960
530 FOR J = 1 TO Q:R = 0:PRINT
540 PRINT"ANALYSIS FOR QUESTION NO.";J
545 PRINT"RESPONSE  COUNT  PERCENT"
550 FOR L = 0 TO C:C(L) = 0:NEXT:M = 0
560 FOR L = 1 TO K:T$ = MID$(Q$(L),J,1)
570 W = VAL(T$):C(W) = C(W)+1:NEXT
610 FOR L = 1 TO C:PRINT L;TAB(11);
620 PRINT C(L);TAB(18);C(L)*100/K;
630 IF LEN(A$) = 0 THEN PRINT:GOTO 660
640 T$ = RIGHT$(STR$(L),1)
650 IF T$ = MID$(A$,J,1) THEN PRINT"C":GOTO 660
655 PRINT
660 NEXT:PRINT"BLANK";TAB(11);C(0);TAB(18);
670 PRINT C(0)*100/K:PRINT P$
680 GET X$: IF X$ = "" THEN 680
690 NEXT J:GOTO 960
700 L = 0:GOSUB 900:IF LEN(A$) = 0 THEN 720
710 PRINT TAB(2);A$;"--KEY"
720 FOR J = 1 TO K
730 PRINT TAB(2);Q$(J);"--NO";J;
740 PRINT
750 L = L+1:IF L < 10 THEN 780
760 L = 0:PRINT P$
770 GET X$:IF X$ = "" THEN 770
780 NEXT:PRINT P$
790 GET X$:IF X$ = "" THEN 790
800 GOTO 960
850 FOR J = 1 TO LEN(T$)
855 IF MID$(T$,J,1) = " " THEN 870
860 IF MID$(T$,J,1) < "1" THEN 880
865 IF MID$(T$,J,1) > C$ THEN 880
870 NEXT:RETURN
880 T$ = "B":RETURN
900 W = Q/10:IF W < 1 THEN 920
910 FOR J = 1 TO W:PRINT TAB(J*10);J;:NEXT:PRINT
920 PRINT TAB(2);
```

```
930 FOR J = 1 TO Q:T$ = STR$(J)
940 PRINT MID$(T$,LEN(T$),1);:NEXT:PRINT:RETURN
960 PRINT:PRINT"NEXT ACTION:"
970 PRINT"1 - ANALYZE QUESTIONS"
980 PRINT"2 - ADD MORE ENTRIES"
990 PRINT"3 - REVIEW DATA ENTERED"
1000 PRINT"4 - END PROGRAM"
1040 INPUT T$:T$ = LEFT$(T$,1)
1050 IF T$ < "1" OR T$ > "4" THEN 1070
1060 ON VAL(T$) GOTO 530,480,700,1100
1070 PRINT E$:GOTO 960
1100 END

READY.
```

EASY CHANGES

1. You can allow for more questions per exam, more choices per question, and more students (or questionnaire respondents). For example, to allow for 25 questions, five choices per question, and 40 students, make this change:

   150 Q=25:C=5:N=40

MAIN ROUTINES

120- 140	Initializes variables.
150- 160	Sets limits for questions, choices, and entries. Allocates arrays.
250- 320	Gets answer key (if any) from operator.
330- 335	Checks legality of answer key.
350- 400	Gets exam data for Kth entry.
430- 475	Scores Kth exam, if applicable.
500- 510	Displays average score, if an exam.
530- 690	Analyzes responses to each question.
700- 800	Displays data entered.
850- 880	Subroutine to check legality of input data.
900- 940	Subroutine to display guide numbers over input data area.
960-1100	Displays choices for next action. Gets response and goes to appropriate routine.

MAIN VARIABLES

E$	Error message.
P$	Message about pressing a key to continue.
Q	Number of questions.
C	Number of choices per question.
C	Array for tallying number of people responding with each choice.
N	Maximum number of entries.
Q$	Array of N strings of entries.
A$	Answer key string (null if not an exam).
C$	String value of highest legal answer choice.
K	Counter of number of exams scored.
R	Number of questions answered right (if exam).
W	Work variable.
J,L,M	Loop variables.
TR	Total right for all entries.
T$	Temporary work string variable.

SUGGESTED PROJECTS

1. Add an option to change the answer key after the data for the exams is entered. This would be useful in case a mistake is found when reviewing the data.
2. Add an option to allow the operator to re-score each of the exams after all are entered, in case some were overlooked at the time of entry.
3. Combine some of the capabilities of the STATS program with this one.
4. Allow the operator to enter a name for each exam paper. This will make it easier to review which person's exam has been entered when option three is used.

SORTLIST

PURPOSE

This program sorts a list of items (words or phrases) into alphabetical order. This is a tedious task to do manually, but your computer can do it in seconds. All you need to do is type in the list of items that need to be sorted.

HOW TO USE IT

Simply type in the list of items that you want sorted, pressing **RETURN** after each one. An item does not have to be single word. You can have embedded spaces, but not commas or colons (unless the entire item is enclosed in quotation marks).

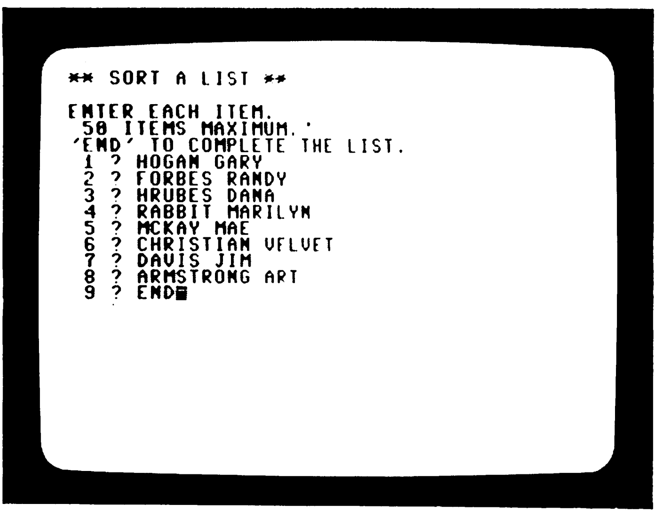
When done with your list, type the word END and press **RETURN**. The program then tells you how many items were entered and begins displaying them in sorted (alphabetical) order. If you have more than 14 items, you will want to use the Commodore 64's capability to stop scrolling the display by pressing **STOP/RUN**. This will stop the program before the first items disappear from the screen. Then you can let the list continue by typing CONT followed by RETURN. If you do not stop the list quickly enough, you can type GOTO 500 after the program ends to display the list again. You may also want to try the Easy Change below that puts more than one item on each line of the display.

This program sorts string data. This means that you can also enter numeric data, but it will not sort numerically the way you probably would want. For example, if you entered the numbers

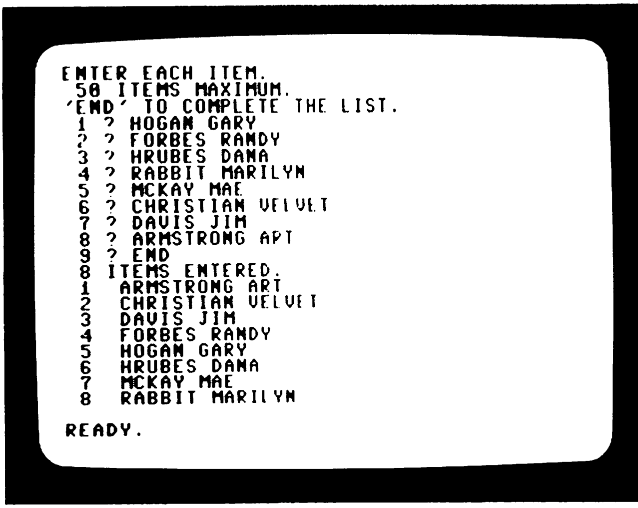
1, 2, 13, and 20, they would be sorted into the sequence 1, 13, 2, 20. The number 13 is sorted ahead of 2 because the first position gets sorted "alphabetically," and 1 comes ahead of 2.

You may find this program useful as shown, or you may want to make use of the technique it uses as part of a larger program. The sorting technique used is called a *straight selection* sort (see the book by Knuth in the bibliography). It has the advantages of being very simply programmed and executing quite quickly for small lists—no more than 30 to 50 or so. This program typically takes about four to seven seconds to sort 30 items, and about eight to thirteen seconds for 40 items, depending on the length and initial sequence of the items.

SAMPLE RUN



The operator enters eight names, and END to end the entry list.



The program displays the alphabetized list, and then ends.

PROGRAM LISTING

```
READY.

100 REM: SORT A LIST
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
130 CLR
140 N = 50:E$ = "END"
150 DIM A$(N):GOSUB 600
160 K = 1
170 IF K > N THEN 250
175 R$ = ""
180 PRINT K;:INPUT R$:IF R$ = "" THEN 180
185 IF R$ = E$ THEN 250
200 A$(K) = R$:K = K+1:GOTO 170
250 K = K-1:IF K > 0 THEN 300
260 PRINT"** NO INPUT TO SORT **"
270 GOTO 160
300 PRINT K;"ITEMS ENTERED."
350 IF K = 1 THEN 500
```

```
360 FOR J = K TO 2 STEP -1
370 R$ = A$(1):F = 1
380 FOR L = 2 TO J
390 IF A$(L) > R$ THEN R$ = A$(L):F = L
400 NEXT:A$(F) = A$(J):A$(J) = R$
410 NEXT
500 FOR J = 1 TO K
510 PRINT J;TAB(4);A$(J)
520 NEXT
550 END
600 PRINT CHR$(147)
610 PRINT"** SORT A LIST **"
620 PRINT
630 PRINT"ENTER EACH ITEM."
640 PRINT N;"ITEMS MAXIMUM."
650 PRINT"'";E$;"' TO COMPLETE THE LIST."
660 RETURN

READY.
```

EASY CHANGES

1. Some simple changes can allow the program to handle more data. You can allow for up to 100 items, where the average item is no more than nine or ten characters long, by making this change:

   140 N = 100:E$ = "END"

   You can have up to about 800 items of the same length with:

   140 N = 800:E$ = "END"

   Be aware that 100 items may take 40 to 60 seconds or more to sort, and 800 will take more than simply eight times that.
2. If you want to sort numbers instead of alphabetic data, make these changes:
   a. Delete the dollar signs in lines 150, 200, 370, 390, 400, and 510.
   b. Insert this line:
      195 R = VAL(R$)
   c. Change the title, if you wish. For example, replace "ITEM" in line 630 with "NUMBER," and "ITEMS" in 640 with "NUMBERS."
3. To reduce the problem of getting only 14 to 16 items on the screen at once, you can try one of these changes:

   510 PRINT J;A$(J);"/";

   or

   510 PRINT A$(J);"/";

This will print multiple items separated by slashes on each line. The second change will also eliminate the item number from the display. Of course, you can separate the items with some character other than a slash if you like.

4. To use some word other than END to indicate the end of the list of items, change line 140. For example, to use DONE:

```
140 N=50:E$="DONE"
```

5. To slow down the display of the sorted list, you can insert:

```
515 FOR L=1 TO 200:NEXT
```

MAIN ROUTINES

130-150	Initializes variables. Displays instructions.
160-300	Inputs items to be sorted.
350-410	Sorts items alphabetically.
500-550	Displays sorted items and ends program.
600-660	Subroutine to display title and instructions.

MAIN VARIABLES

N	Maximum number of items that can be entered.
E$	Word to end entry of items.
A$	Array of items to be sorted (in place).
K	Count of number of items actually entered.
R$	Reply from operator. Also a work string variable.
J,L,F	Work and subscript variables.

SUGGESTED PROJECTS

1. Replace the sorting technique with one that is more efficient for large numbers of items. Knuth's and Gruenberger's books (see bibliography) both have discussions of alternatives.
2. Give the program the capability to add or change some items after the list has been sorted.
3. Add an option to allow the sorted list to be saved on cassette so it can be loaded into another program when needed.

Section 2

Educational Programs

INTRODUCTION TO EDUCATION PROGRAMS

Education is one area where computers are certain to have more and more impact. Though a computer cannot completely replace a human teacher, the machine does have certain advantages. It is ready anytime you are, allows you to go at your own pace, handles rote drill effortlessly, and is devoid of any personality conflicts.

With a good software library, the Commodore 64 can be a valuable learning center in the school or at home. Here are seven programs to get you started.

Mathematics is certainly a "natural" subject for computers. NUMBERS is designed for pre-school children. While familiarizing youngsters with computers, it provides an entertaining way for them to learn numbers and elementary counting. ARITHMETIC is aimed at older, grade school students. It provides drill in various kinds of math problems. The child can adjust the difficulty factors, allowing the program to be useful for several years.

By no means is the Commodore 64 restricted to mathematical disciplines. We include two programs designed to improve your word skills. VOCAB will help you expand your vocabulary. TACHIST turns the computer into a reading clinic, helping you to improve your reading speed.

With the proper programs, the computer can teach you specific subjects. If you've ever wanted to learn International Radio Code, HAMCODE will instruct and then drill you. Many

of us feel uncomfortable becoming familiar with the increasingly prevalent metric system. METRIC is the answer to this.

But, what about software that you can customize to help you learn a subject of your choice? FLASHCARD allows you to create your own "computer flashcards." Then you can drill yourself until you get it right.

ARITHMETIC

PURPOSE

ARITHMETIC provides mathematics drills for grade school children. The student can request problems in addition, subtraction, or multiplication from the program. Also, he or she may ask that the problems be easy, medium, or hard. The program should be useful to a child over an extended period of time. He can progress naturally to a harder category of problems when he begins to regularly perform well at one level. The difficulty and types of problems encompass those normally encountered by school children between the ages of six and ten.

The problems are constructed randomly within the constraints imposed by the degree of difficulty selected. This gives the student fresh practice each time the program is used. After entering answers, he is told whether he was right or wrong. The correct answers are also displayed.

HOW TO USE IT

First, the student must indicate what type of problem he wishes to do. The program requests an input of **1, 2,** or **3** to indicate addition, subtraction, or multiplication, respectively. It then asks whether easy, medium, or hard problems are desired. Again an input of **1, 2,** or **3** is required.

Now the screen will clear and four problems of the desired type will be displayed. The user now begins to enter his answers to each problem.

A question mark is used to prompt the user for each digit of the answer, one digit at a time. This is done moving right to left, the way arithmetic problems are naturally solved.

To start each problem, the question mark will appear in the spot for the rightmost (or units column) digit of the answer. When the key for a digit from 0-9 is pressed, that digit will replace the question mark on the screen. The question mark moves to the immediate left waiting for a digit for the "tens" column.

Digits are entered in this right to left manner until the complete answer has been input. Then the **RETURN** key must be pressed. This will end the answer to the current problem and move the question mark to begin the answer for the next question.

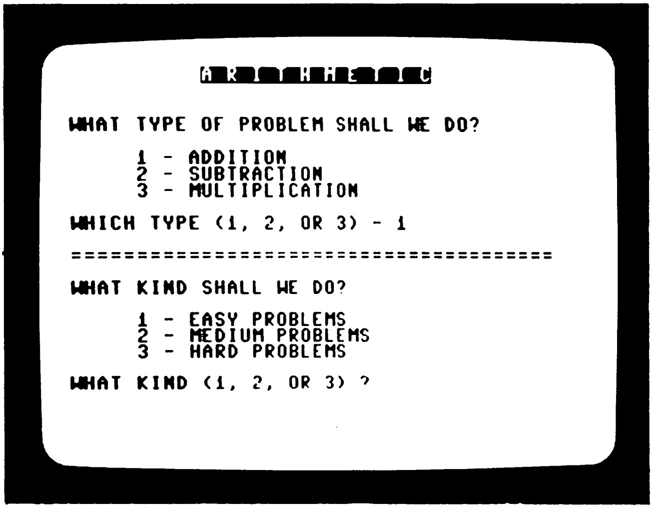
If the **RETURN** key is pressed to begin a problem, an answer of zero is assumed intended. No problems created by this program have answers of more than three digits. If a four-digit answer is given, the program will accept the answer, but then go immediately to the next problem. Answers to the problems are never negative.

The program will display the correct answers to the four problems on the screen after the student has entered his four answers. The message "RIGHT!" or "WRONG!" will also be displayed below each problem.

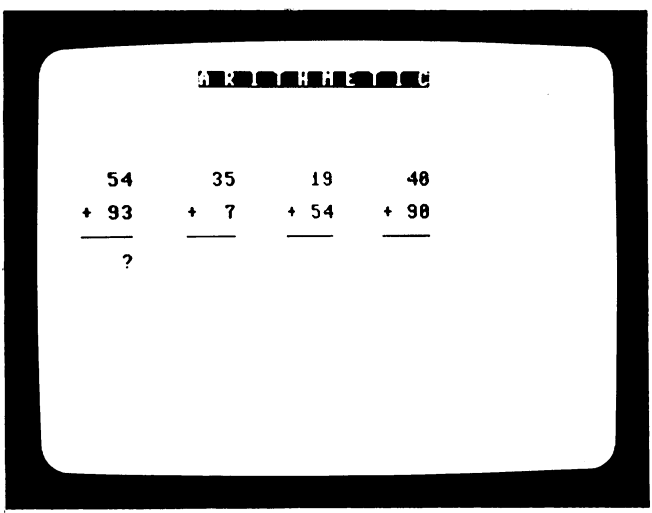
Then the message "HIT ANY KEY TO CONTINUE" will be displayed. After the key is pressed, a new set of four problems of the same type will be presented.

This continues until twenty problems have been worked. The program then shows what the student's performance has been. This is expressed as the number of problems solved correctly and also as the percentage of problems solved correctly.

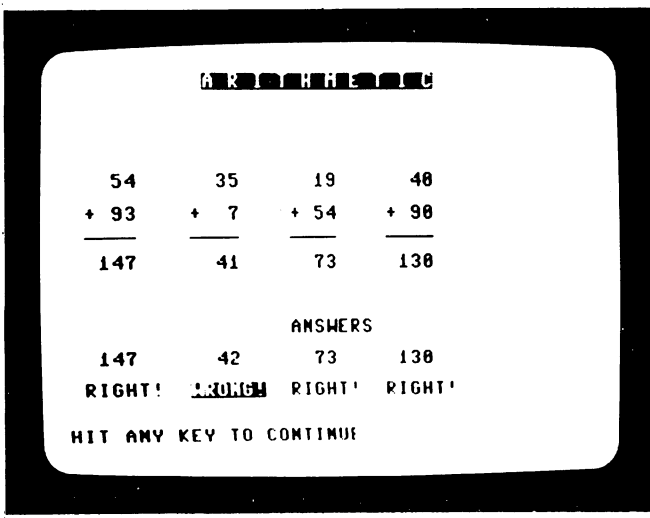
SAMPLE RUN



The operator chooses to do hard addition problems.



The initial set of 4 problems is presented. With a question mark, the program prompts the operator for the answer to the first problem.



The operator has entered his or her four answers. The program displays the correct answers and indicates whether or not each problem was solved correctly. The program waits for the operator to hit any key in order to continue with the next set of four problems.

PROGRAM LISTING

```
READY.

100 REM: ARITHMETIC
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
140 S = 1024
150 ND=0:R=RND(-TI)
160 DIM A(4),B(4),C(4),G(4)
170 NP=20
180 GOSUB 910
200 PRINT:PRINT
205 PRINT"WHAT TYPE OF PROBLEM SHALL WE DO?":PRINT
210 PRINT TAB(5);"1 - ADDITION"
220 PRINT TAB(5);"2 - SUBTRACTION"
230 PRINT TAB(5);"3 - MULTIPLICATION"
240 PRINT:PRINT"WHICH TYPE (1, 2, OR 3) ?";
```

```
250 GET R$:T=VAL(R$):IF T<1 OR T>3 THEN 250
260 PRINT CHR$(157);"-";T
270 PRINT:FOR J=1 TO 39:PRINT"=";:NEXT
275 PRINT:PRINT:PRINT"WHAT KIND SHALL WE DO?"
280 PRINT:PRINT TAB(5);"1 - EASY PROBLEMS"
290 PRINT TAB(5);"2 - MEDIUM PROBLEMS"
300 PRINT TAB(5);"3 - HARD PROBLEMS"
310 PRINT:PRINT "WHAT KIND (1, 2, OR 3) ?";
320 GET R$:D=VAL(R$):IF D<1 OR D>3 THEN 320
330 PRINT CHR$(157);"-";D
350 ON D GOTO 360,370,390
360 GOSUB 940:GOSUB 920:GOSUB 930:GOTO 400
370 GOSUB 940:GOSUB 930:IF T=3 THEN GOSUB 960
375 GOSUB 920:GOTO 400
380 IF T<>3 THEN GOSUB 950:GOSUB 920:GOTO 400
390 GOSUB 950:GOSUB 920:GOSUB 930
395 IF T=3 THEN GOSUB 940:GOSUB 930
400 IF T<>2 THEN 450
410 FOR J=1 TO 4
420 IF B(J)>C(J) THEN R=C(J):C(J)=B(J):B(J)=R
430 NEXT
450 GOSUB 1000:GOSUB 910
600 Y=12:FOR J=1 TO 4:X=-4+J*8:GOSUB 1100:NEXT
610 FOR K=1 TO 4:X=-4+K*8:GOSUB 800:G(K)=N:NEXT
620 X=17:Y=16:GOSUB 1200:PRINT"ANSWERS"
630 Y=18:FOR J=1 TO 4:X=-4+J*8:GOSUB 1400:NEXT
640 Y=20:FOR J=1 TO 4:X=-7+J*8:GOSUB 1200
650 IF A(J)<>G(J) THEN PRINT CHR$(18);"WRONG!"
655 IF A(J)<>G(J) THEN GOTO 670
660 PRINT"RIGHT!":NR=NR+1
670 NEXT:FOR K=1 TO 9:GET R$:NEXT
680 PRINT:PRINT:PRINT"HIT ANY KEY TO CONTINUE"
690 GET R$:IF R$="" THEN 690
700 FOR J=1 TO 10:GET R$:NEXT
710 ND=ND+4:IF ND<NP THEN GOSUB 910:GOTO 350
720 GOSUB 1500
730 END
800 N=0:M=1:FOR J=1 TO 10:GET R$:NEXT
810 P=63:GOSUB 900
820 GET R$:IF R$="" THEN 820
825 A=ASC(R$)
827 IF A=13 AND M=1 THEN P=48:GOSUB 900:RETURN
830 IF A=13 THEN P=32:GOSUB 900:RETURN
840 V=VAL(R$):IF V=0 AND A<>48 THEN 820
850 P=48+V:GOSUB 900:N=N+M*V:M=M*10
860 IF M>1000 THEN RETURN
870 X=X-1:GOTO 810
900 POKE S+X+40*Y,P:RETURN
910 PRINT CHR$(147);CHR$(13);TAB(10);CHR$(18);
915 PRINT "A R I T H M E T I C":RETURN
920 FOR K=1 TO 4:C(K)=INT(RND(1)*(H-L+1))+L
925 NEXT:RETURN
```

```
930 FOR K=1 TO 4:B(K)=INT(RND(1)*(H-L+1))+L
935 NEXT:RETURN
940 H=9:L=0:RETURN
950 H=99:L=0:RETURN
960 H=25:L=1:RETURN
1000 ON T GOTO 1010,1020,1030
1010 FOR J=1 TO 4:A(J)=B(J)+C(J):NEXT:RETURN
1020 FOR J=1 TO 4:A(J)=C(J)-B(J):NEXT:RETURN
1030 FOR J=1 TO 4:A(J)=C(J)*B(J):NEXT:RETURN
1100 GOSUB 1200:CU=5:CL=2:GOSUB 1300
1110 IF C(J)<10 THEN PRINT CHR$(32);
1120 PRINT C(J):GOSUB 1200:CU=3:CL=3:GOSUB 1300
1125 IF T=1 THEN PRINT CHR$(43);
1130 IF T=2 THEN PRINT CHR$(45);
1140 IF T=3 THEN PRINT CHR$(214);
1150 IF B(J)<10 THEN PRINT CHR$(32);
1160 PRINT B(J):GOSUB 1200:CU=2:CL=3:GOSUB 1300
1165 FOR K=1 TO 4:PRINT CHR$(164);
1170 NEXT:RETURN
1200 PRINT CHR$(19);
1205 FOR K=1 TO X:PRINT CHR$(29);:NEXT
1210 FOR K=1 TO Y:PRINT CHR$(17);:NEXT:RETURN
1300 FOR K=1 TO CU:PRINT CHR$(145);:NEXT
1310 FOR K=1 TO CL:PRINT CHR$(157);:NEXT:RETURN
1400 GOSUB 1200:CL=1:IF A(J)>9 THEN CL=2
1410 IF A(J)>99 THEN CL=3
1420 IF A(J)>999 THEN CL=4
1430 FOR K=1 TO CL:PRINT CHR$(157);:NEXT
1440 PRINT A(J):RETURN
1500 GOSUB 910:PRINT:PRINT
1510 PRINT"YOU GOT";NR;" RIGHT
1520 PRINT"OUT OF";NP;" PROBLEMS
1530 P=NR/NP*100:
1535 PRINT
1540 PRINT"THAT'S";P;" PERCENT CORRECT":RETURN

READY.
```

EASY CHANGES

1. The program currently does twenty problems per session. you can change this number by altering the variable NP in line 170. For example,

   170 NP = 12

   will cause the program to do only twelve problems per session. The value of NP should be kept a positive multiple of four.

2. Zero is currently allowed as a possible problem operand. If you do not wish to allow this, change lines 940 and 950 to read as follows:

```
940 H=9:L=1:RETURN
950 H=99:L=1:RETURN
```

MAIN ROUTINES

150- 180	Initializes constants.
200- 330	Asks operator for type of problems desired.
350- 450	Sets A, B, C arrays, clears screen.
600- 730	Mainline routine—displays problems, gets operator's answers, displays correct answers and user's performance.
800- 870	Subroutine to get and display user's answers.
900	Subroutine to poke byte P into screen position X,Y.
910- 915	Subroutine to clear screen and display title.
920- 935	Subroutine to set B,C arrays.
940- 960	Subroutine to set L,H.
1000-1030	Subroutine to calculate array A from arrays B, C.
1100-1170	Subroutine to display problems.
1200-1210	Subroutine to move cursor to screen position X,Y.
1300-1310	Subroutine to move cursor CO lines down and CL spaces left.
1400-1440	Subroutine to display the correct answers.
1500-1540	Subroutine to display operator's performance.

MAIN VARIABLES

NP	Number of problems to do in the session.
ND	Number of problems done.
NR	Number of correct answers given.
C,B,A	Arrays of top operand, bottom operand, and correct answer to each problem.
N	Operator's answer to current problem.
G	Array of operator's answers.
T	Type of problems requested (1 = addition, 2 = subtraction, 3 = multiplication).
D	Kind of problem requested (1 = easy, 2 = medium, 3 = hard).

H,L	Highest, lowest integers to allow as problem operands.
M	Answer column being worked on.
R$	Operator's input character.
V	Value of R$.
A	ASCII value of R$.
X,Y	Horizontal, vertical screen position of cursor.
S	Starting address of CRT memory area.
R	Work variable.
J,K	Loop indices.
P	Poke character, also percentage correct.
CU,CL	Number of positions to move cursor up, left.

SUGGESTED PROJECTS

1. Keep track of problems missed and repeat them quickly for additional practice.
2. No negative operands or answers are currently allowed. Rewrite the problem generation routines and the operator's answer routines to allow the possibility of negative answers.
3. The answers are now restricted to three-digit numbers. However, the program will work fine for four-digit numbers if the operands of the problems are allowed to be large enough. Dig into the routines at lines 350-450 and 940-960. See how they work and then modify them to allow possible four-digit answers.
4. The operator cannot currently correct any mistakes he makes while typing in his answers. Modify the program to allow him to do so.
5. Modify the program to allow problems in division.

FLASHCARD

PURPOSE

There are certain things that the human mind is capable of learning only through repetition. Not many people can remember the multiplication tables after their first exposure, for example. The same applies to learning the vocabulary of a foreign language, the capital cities of the fifty states, or famous dates in history. The best way to learn them is to simply review them over and over until you have them memorized.

A common technique for doing this involves the use of flashcards. You write one half of the two related pieces of information on one side of a card, and the other half on the other side. After creating a set of these cards, you can drill yourself on them over and over until you always remember what's on the other side of each card.

But why waste precious natural resources by using cards? Use your computer instead. This program lets you create flashcards, drill using them, and save them on cassette tape for later review.

HOW TO USE IT

The program gives you seven options. The first time you run it, you'll want to enter new flashcards, so you should reply with number 1.

To create the cards, the program asks you for each side of each flashcard, one at a time. First enter side one of the first card, and so on. As you enter the data, be careful not to use any commas or colons unless the entire expression is enclosed in quotation marks.

At any time, you can enter the keyword "*BACK" instead of side one to correct an erroneous entry. This causes the program to back up and ask you for the previous card again.

As the program is currently written, you must enter at least five flashcards, and no more than twenty-five. We will show you how to change these limits in the "Easy Changes" section.

When you have entered all the flashcards you want, enter "*END" instead of side one of the next card. This puts the program back into "command" mode to ask you what to do next. If you want to quiz yourself on the cards you just entered, respond with the number 4.

The program flashes one side of one card on the screen for you. Both are chosen at random—the side and the card. Your job is to respond with the other side. If you enter it correctly, the program says "RIGHT!" If not, it tells you the correct response. In either event, the program continues by picking another side and card at random. This continues until you respond with "*END", which tells the program you do not want to drill any more. It will then tell you how many you got right out of the number you attempted, as well as the percentage, and then return to command mode.

During the drill sequence, by the way, the program will not repeat a card that was used in the previous two questions (i.e., one less than the minimum number of cards you can enter).

To save a set of flashcards on cassette, use option number 3. The program will tell you to put the cassette into position and then enter a name for the file. You should give it a good descriptive name in order to remember what kind of flashcards they are in the future. Be sure to write the name on the cassette, too. After the flashcards have been copied to the cassette, the program will say "DONE" and return to the command mode.

The other commands are easily understood, so we will just explain them briefly. A little experimentation will show you how they work.

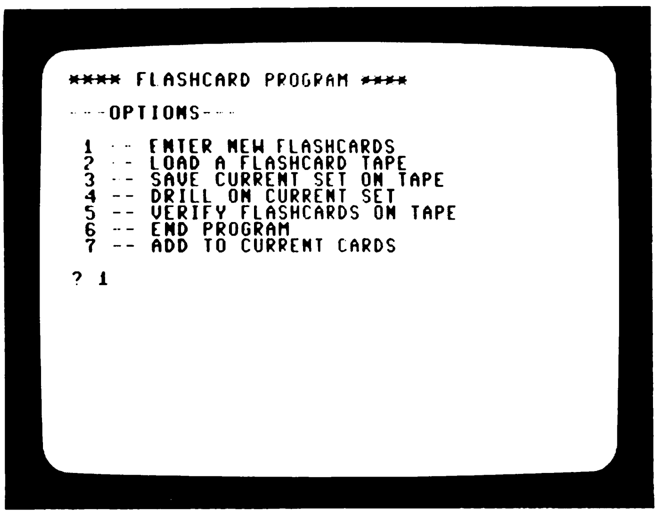
Command number 2 is used to load a flashcard tape that has been previously saved. The program asks for the name of the file, so it can scan the cassette until it finds the one you asked for. If you don't care or don't know the name of the file, you can load the first file that is found on the cassette by entering a null string for the name (two consecutive double quote marks).

Command number 5 lets you verify that the set of flashcards just saved on tape is okay. You do not have to enter the name of the file—the same name is used that was used to save the file.

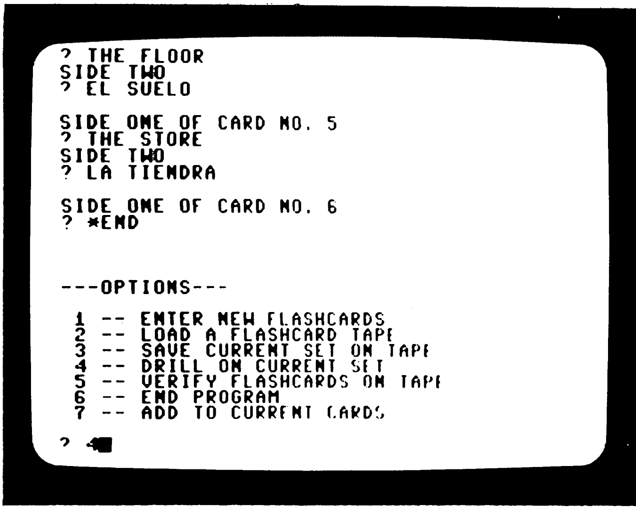
Command number 6 ends the program.

Command number 7 allows you to add more flashcards to those currently in memory.

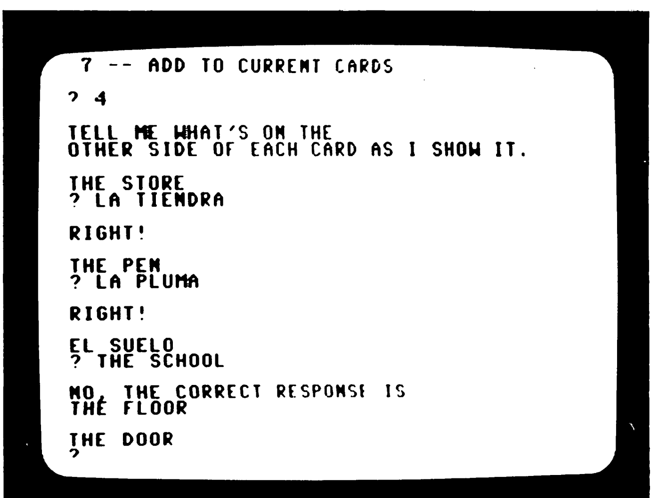
SAMPLE RUN



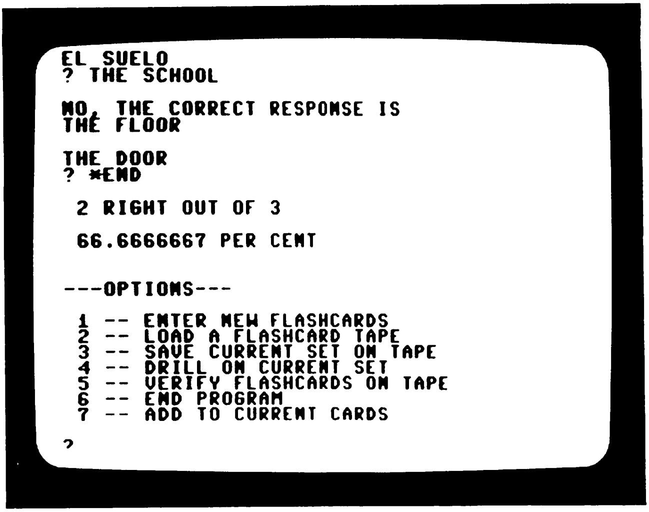
The program begins by showing a menu of options. The operator selects option 1 to start entering the front and back of the flash cards.



The operator responds with *END after he has entered five cards. He then selects option 4 to start the drill on the current set.



The operator gets the first two answers right, but misses the third. The program responds by giving the correct answer.



The operator ends the drill and the program shows the number and percentage of correctly answered questions. The options for continuing are displayed and the program waits for the operator's choice.

PROGRAM LISTING

```
READY.

100 REM: FLASHCARD
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
130 L=25:M=5
140 DIM F$(L),B$(L),P(M-1)
150 R=RND(-TI)
160 PRINT CHR$(147)
170 PRINT"**** FLASHCARD PROGRAM ****"
180 GOTO 2000
190 K=1:W=0:C=0:PRINT
200 PRINT"SIDE ONE OF CARD NO.";K:INPUT F$(K)
210 IF LEFT$(F$(K),4)="*END" THEN 280
220 IF LEFT$(F$(K),5)<>"*BACK" THEN 230
222 K=K-1:IF K<1 THEN K=1
225 PRINT:PRINT"BACKING UP":GOTO 200
230 PRINT"SIDE TWO":INPUT B$(K)
240 IF LEFT$(B$(K),5)="*BACK" THEN 225
250 PRINT
```

```
260 K=K+1:IF K<=L THEN 200
270 PRINT"THAT'S ALL THERE'S ROOM FOR."
280 PRINT:PRINT:K=K-1:GOTO 2000
290 IF K>=M THEN 310
300 PRINT"THAT'S ONLY";K;"CARDS. MINIMUM IS";M
305 GOTO 2000
310 PRINT:PRINT"TELL ME WHAT'S ON THE"
320 PRINT"OTHER SIDE OF EACH CARD AS I SHOW IT."
330 PRINT
340 R=INT(K*RND(1))+1
350 FOR J=0 TO M-2
360 IF P(J)=R THEN 340
370 NEXT
390 J=RND(1):IF J>.5 THEN 420
400 PRINT F$(R):C$=B$(R)
410 GOTO 430
420 PRINT B$(R):C$=F$(R)
430 INPUT R$
440 IF LEFT$(R$,4)="*END" THEN 600
450 PRINT
460 IF R$=C$ THEN 500
470 PRINT"NO, THE CORRECT RESPONSE IS"
480 PRINT C$
490 W=W+1:GOTO 520
500 PRINT"RIGHT!"
510 C=C+1
520 FOR J=1 TO M-2
530 P(J-1)=P(J):NEXT
540 P(M-2)=R
550 PRINT
560 GOTO 340
600 GOSUB 1500
610 GOTO 2000
700 IF K<1 THEN 1800
710 PRINT:PRINT"PUT CASSETTE INTO POSITION.":PRINT
720 INPUT"NAME FOR FILE";R$
730 OPEN 7,1,1,R$
740 FOR J=1 TO K
750 PRINT#7,F$(J):PRINT#7,B$(J)
770 NEXT
780 CLOSE 7
790 PRINT:PRINT"DONE"
800 N$=R$
810 GOTO 2000
900 PRINT:PRINT"REPOSITION TAPE AHEAD OF FILE"
910 PRINT"THEN PRESS ANY KEY."
920 GET R$:IF R$="" THEN 920
930 OPEN 7,1,0,N$
940 E=0:FOR J=1 TO K
950 INPUT#7,R$:IF 64 AND ST THEN 1030
960 INPUT#7,T$
```

```
965 PRINT R$;"-";T$
970 IF 64 AND ST THEN 1000
980 IF R$<>F$(J) OR T$<>B$(J) THEN E=1
990 NEXT
1000 CLOSE 7
1010 PRINT:PRINT"DONE"
1020 IF E=0 THEN PRINT"O.K.":GOTO 2000
1030 PRINT"NO GOOD":GOTO 2000
1150 INPUT"NAME OF TAPE FILE";N$
1170 PRINT:PRINT"PUT TAPE INTO POSITION"
1180 PRINT"THEN PRESS ANY KEY."
1190 GET R$:IF R$="" THEN 1190
1200 OPEN 7,1,0,N$
1210 K=1:W=0:C=0
1220 INPUT#7,R$
1230 IF 64 AND ST THEN 1290
1240 INPUT#7,T$
1250 PRINT R$;"-";T$
1270 F$(K)=R$:B$(K)=T$
1280 K=K+1:IF K<L THEN 1220
1290 K=K-1:PRINT:PRINT"LOADED";K;"CARDS"
1300 CLOSE 7
1310 PRINT:GOTO 2000
1500 PRINT
1505 IF C+W=0 THEN RETURN
1510 PRINT C;"RIGHT OUT OF";C+W
1520 PRINT:PRINT C*100/(C+W);"PER CENT"
1530 PRINT
1540 RETURN
1800 PRINT:PRINT"NO CARDS YET"
1810 GOTO 2000
2000 PRINT:PRINT"---OPTIONS---":PRINT
2010 PRINT" 1 -- ENTER NEW FLASHCARDS"
2020 PRINT" 2 -- LOAD A FLASHCARD TAPE"
2030 PRINT" 3 -- SAVE CURRENT SET ON TAPE"
2040 PRINT" 4 -- DRILL ON CURRENT SET"
2050 PRINT" 5 -- VERIFY FLASHCARDS ON TAPE"
2060 PRINT" 6 -- END PROGRAM"
2065 PRINT" 7 -- ADD TO CURRENT CARDS"
2070 PRINT:INPUT R$
2080 IF R$="1" THEN 190
2090 IF R$="2" THEN 1150
2100 IF R$="3" THEN 700
2110 IF R$="4" THEN 290
2120 IF R$="5" THEN 900
2130 IF R$="6" THEN END
2135 IF R$="7" THEN 260
2140 PRINT"ILLEGAL":GOTO 2000


READY.
```

EASY CHANGES

1. Change the limits of the number of flashcards that can be entered by altering line 130. L is the upper limit and M is the minimum. Do not make M much larger than about ten or so, or you will slow down the program and use more memory than you might want.
2. If you want to use some keywords other than "*END" and "*BACK", substitute whatever you like in lines 210, 220, 240 and 440. Be sure you use expressions that are the same length as these two, however. If not, you will also need to change the last number just before each occurrence of the expression to correspond with the length.
3. To cause the program to always display side one of the flashcards (and ask you to respond with side two), insert this line:

   380 GOTO 400

   To cause it to always display side two, insert this:

   380 GOTO 420
4. To eliminate the "echoing" on the screen of a tape file being loaded, remove line 1250. To do the same for a tape being verified, remove line 965.

MAIN ROUTINES

130- 180	Initializes variables. Creates arrays. Displays title and options.
190- 280	Accepts flashcards entered by operator.
290- 610	Drills operator on flashcards in memory.
700- 810	Saves flashcards on cassette file.
900-1030	Verifies that flashcards on tape are the same as those in memory.
1150-1310	Loads flashcards from cassette file into memory.
1500-1540	Subroutine to display number right and attempted during drill.
1800-1810	Displays error message if operator tries to save flashcards on cassette before any are entered.
2000-2140	Displays options and analyzes response. Branches to appropriate routine.

MAIN VARIABLES

L	Upper limit of number of flashcards that can be entered.
M	Minimum number of flashcards that can be entered.
R	Subscript of random flashcard chosen during drill.
K	Number of flashcards entered.
W	Number of wrong responses.
C	Number of correct responses.
F$	Array containing front side of flashcards (side 1).
B$	Array containing back side of flashcards (side 2).
P	Array containing subscripts of M – 1 previous flashcards during drill.
J	Loop and subscript variable.
C$	The correct response during drill.
R$	Response from operator. Also temporary string variable.
N$	Name of cassette file.
E	Error flag—set to 1 if error occurs on cassette.
T$	Temporary string variable.

SUGGESTED PROJECTS

1. Modify the program for use in a classroom environment. You might want to allow only command 2 to be used (to load a cassette tape), and then immediately go into "drill" mode for some fixed number of questions (maybe 20 or 50).

HAMCODE

PURPOSE

At some time in your life you have undoubtedly heard the sound of "Morse Code." The familiar sound of dots and dashes is one that we have nearly all come in contact with at some time or other. Amateur radio operators ("hams") have to learn Morse code to obtain a license to operate an amateur radio station.

This program helps teach you what is officially called Continental Code, or sometimes referred to (ambiguously) as the International Morse Code, as used for ham radio. This is a little confusing, since the so-called *International* Morse Code, although similar, is *not* the same as Morse Code, which is still in use only for some types of land line transmissions. The code that nearly everyone uses anymore is Continental Code, so we picked the name HAMCODE for this chapter to try to be most descriptive of its use.

HOW TO USE IT

The program begins by displaying its title and sounding it in code for you. It then shows you five options to choose from. To learn the code, you will be selecting different options to enable you to learn each character and become proficient at understanding groups of characters.

The first option teaches you each character. All you do is press any key on the computer's keyboard, and the program sounds the code of that character for you. As the code is sounded, the dots and dashes for it are displayed to help you

both visualize and hear the code together. Be sure you have the sound on your TV set at a comfortable volume for hearing the code.

It's up to you to decide which characters you want to learn first. Most people will find it easiest to learn only a few each day (maybe three or four), and drill on them until they can be recognized immediately. Then add a few more characters the next day. Start with the alphabet, then move on to the numbers and punctuation characters. Keeping each session short is a good idea—half an hour is about right. Doing two or three short sessions each day is better than doing one long one.

Some keys on the computer keyboard do not have a code assigned to them. If you press one of the "illegal" keys, a distinctive low beep is sounded to let you know. To end the character-learning option, press the **CLR/HOME** key. This causes the five options to be displayed for you again.

After you have learned a few letters of the alphabet, you may want to try listening to groups of letters (words or phrases). This is done with option two. Simply enter one or more characters and press the **RETURN** key. The program will respond by sounding the code of the entire phrase at a rate of about 12 words per minute (five characters comprise an average word). The Easy Changes section shows how to change the speed, either faster or slower.

If the phrase has multiple words, they are separated by light blue blocks on the color video display. As the program is currently written, you should limit the length of your phrases to no more than 40 characters. If you want to include any colons or commas in the phrase, you have to enclose the phrase in quotation marks. To hear a phrase a second time, simply press **RETURN** and it will be repeated.

To end option two, enter the word END as your phrase. Once again, this causes the five options to be displayed.

Once you have learned all the characters, you should try option three to quiz yourself on them. Option three randomly picks a character, sounds it for you, and waits for you to press the key of that character. There is no need to press the **RETURN** key. If you press the right key, the program tells you so and picks another random character.

If you press the wrong key, the program tells you what character it was and then sounds it for you again. You have to

respond with the right answer before the program will pick a new character. This helps reinforce the correct answers. To end option three, press the **CLR/HOME** key.

Option four quizzes you on groups of characters which have been chosen at random. As currently written, groups of five characters are used, but the Easy Changes section shows how to make the program use other lengths.

After the five characters are sounded, enter the corresponding five characters and press the **RETURN** key. As with option three, the program tells you if you were right or wrong. If wrong, it tells you the correct answer and sounds the same characters for you again to make you enter them correctly. As with option two, if there are any colons or commas included in the group of characters, you must enclose the entire group of characters within quotation marks. And again, if you simply press the **RETURN** key, the phrase will be repeated for you.

The last option, option five, ends the program.

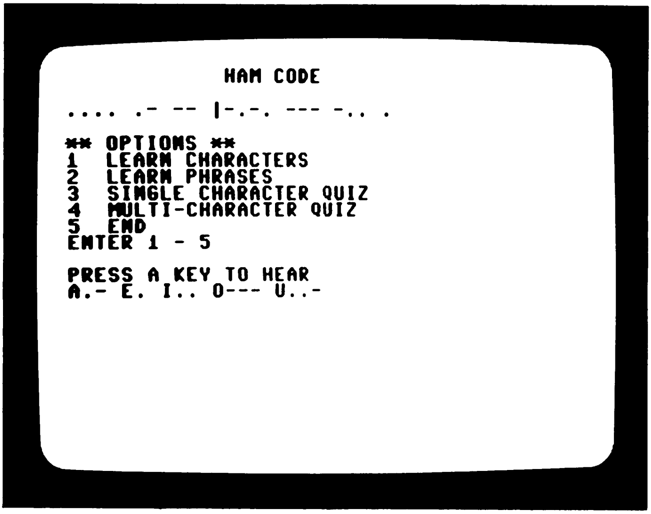
A few characters are not included in this program; these will have to be learned through other means. In all cases but two, this is because there are no ASCII characters on the keyboard to correspond with them (e.g., wait, double dash, error). The two exceptions are the quotation mark and the right parenthesis.

As mentioned above, quotation marks are used by BASIC to enclose a string of characters being entered by the operator. Since this would make it very awkward to include the quotation mark character in the program, and because the other characters are more important to learn, it has been omitted.

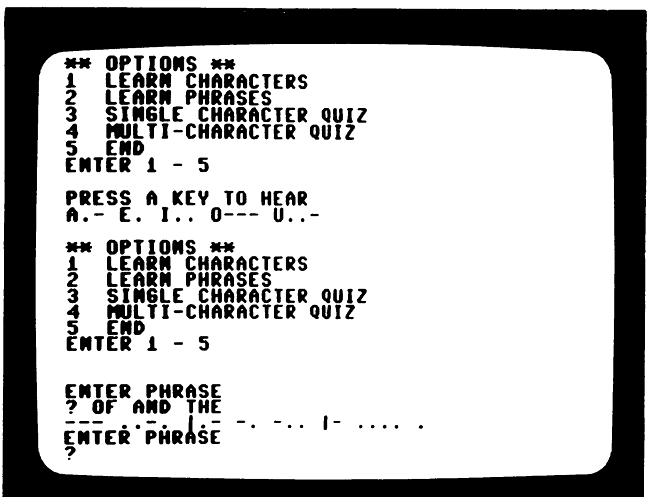
The right and left parenthesis are both supposed to use the same code. To avoid ambiguity in having you figure out whether the program was asking for the left or right parenthesis during the quiz options, we simply decided to treat the right one as an illegal character and thereby allow you to always respond with the left one.

Please be *very* careful when entering this program into your computer, especially for lines 3010 through 3130. If you make a mistake in typing the dots, dashes, commas, and X's, the program either will not work (Out of Data error, most likely), or you will teach yourself the wrong code! Be sure that you compare your results against the Sample Run photos to be sure that your codes look the same as ours.

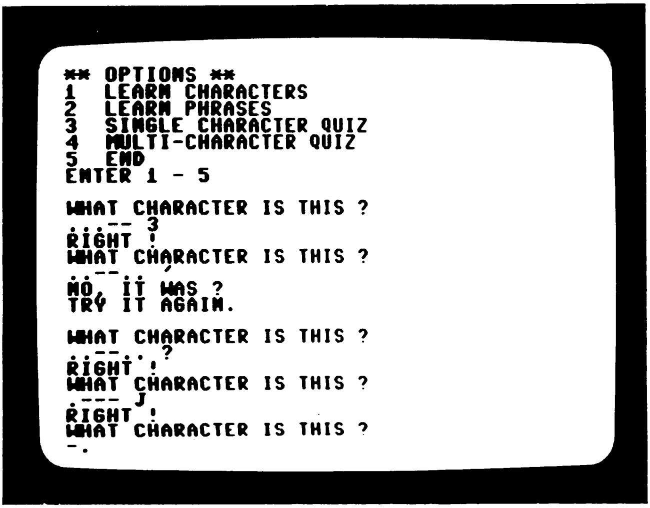
SAMPLE RUN



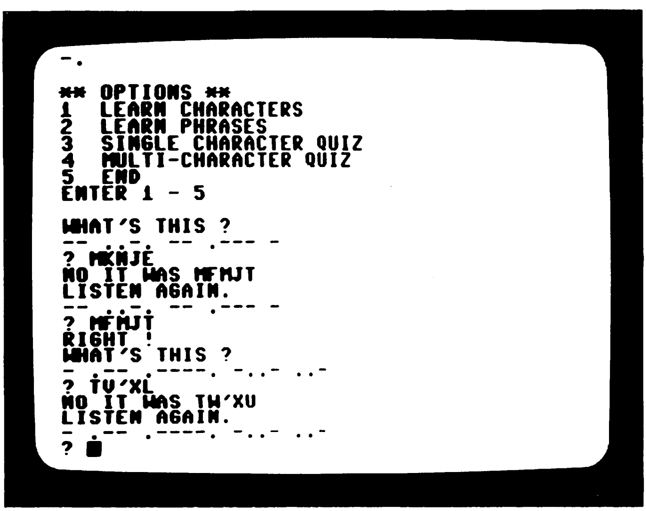
The program displays its title (both alphabetically and in code) and displays its options. The operator picks the first option and begins learning the vowels.



The operator proceeds to option 2, and begins by drilling on some common short words.



Next the operator tries option 3, to be quizzed on individual characters. The first response is correct, but the next is not. The program repeats it to force the operator to respond correctly before going on to the next character.



Finally, the operator asks for option 4, to test himself on random five character groups. After a mistake in his response for the first group, he replies correctly when it is repeated.

PROGRAM LISTING

```
READY.

100 REM: HAM CODE
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
115 POKE 54296,15: REM SET VOLUME
120 GOTO 2000
130 W = ASC(R$)-39:IF W < 0 OR W > 51 THEN 220
140 T$ = C$(W):IF T$ >"/" THEN 220
150 FOR J = 1 TO LEN(T$):W$ = MID$(T$,J,1)
160 D = 220:LE = 220
165 IF W$ = "." THEN D = 100:LE = 100
170 PRINT W$;
180 POKE 54277,128:POKE 54278,D
182 POKE 54273,HI:POKE 54272,LO
184 POKE 54276,17
190 FOR D = 1 TO LE:NEXT
192 POKE 54277,0:POKE 54278,0
194 POKE 54273,0:POKE 54272,0
196 POKE 54276,0
200 NEXT:FOR J = 1 TO 60*T:NEXT
210 PRINT" ";:RETURN
250 FOR K = 1 TO LEN(P$):R$ = MID$(P$,K,1)
260 IF ASC(R$) = 32 THEN 280
270 GOSUB 130:NEXT:RETURN
280 PRINT CHR$(098);:FOR J = 1 TO 90:NEXT
290 NEXT:PRINT:RETURN
300 PRINT"PRESS A KEY TO HEAR"
310 GET R$:IF R$ = "" THEN 310
320 W = ASC(R$):IF W = 19 THEN 2100
330 IF W < 39 OR W > 90 THEN 350
340 IF C$(W-39) <> "X" THEN 360
350 GOSUB 2500:REM ERROR
355 GOTO 310
360 PRINT R$;:GOSUB 130:GOTO 310
400 PRINT:PRINT"ENTER PHRASE"
410 INPUT P$
420 IF LEN(P$) = 0 THEN P$ = L$
430 IF P$ = "END" THEN 2100
440 GOSUB 250:L$ = P$
450 GOTO 400
500 GOSUB 900
510 GOSUB 910
520 PRINT"WHAT CHARACTER IS THIS ?"
540 GOSUB 130
550 GET T$:IF T$ = "" THEN 550
560 IF ASC(T$) = 19 THEN 2100
```

```
570 IF ASC(T$) = 13 THEN 540
580 PRINT T$:IF T$ = R$ THEN 620
585 GOSUB 2500
590 PRINT"NO, IT WAS ";R$
600 PRINT"TRY IT AGAIN.":PRINT
610 GOTO 510
620 PRINT"RIGHT !":GOTO 500
700 PRINT"WHAT'S THIS ?"
710 P$ = "":FOR J = 1 TO N
720 GOSUB 900
730 P$ = P$ + R$:NEXT
740 GOSUB 920:GOSUB 250:PRINT
750 INPUT T$:IF T$ = "" THEN 740
760 IF T$ = "END" THEN 2100
770 IF T$ = P$ THEN PRINT"RIGHT !":GOTO 700
775 GOSUB 2500
780 PRINT"NO IT WAS ";P$
790 PRINT"LISTEN AGAIN."
800 GOTO 740
900 R = INT(RND(1)*52)-1
905 IF C$(R) = "X" THEN 900
910 R$ = CHR$(R+39):RETURN
920 FOR J = 1 TO 800:NEXT:RETURN
950 END
2000 CLR
2010 DIM C$(51)
2020 FOR J = 0 TO 51:READ C$(J):NEXT
2030 T = 5:HI = 34:LO = 75
2050 PRINT CHR$(147):P$ = "HAM CODE"
2060 PRINT TAB(12);P$:PRINT
2070 GOSUB 250
2080 N = 5:L$ = CHR$(32)
2100 PRINT
2110 PRINT:PRINT"** OPTIONS **"
2120 PRINT"1  LEARN CHARACTERS"
2130 PRINT"2  LEARN PHRASES"
2140 PRINT"3  SINGLE CHARACTER QUIZ"
2150 PRINT"4  MULTI-CHARACTER QUIZ"
2160 PRINT"5  END"
2190 PRINT"ENTER 1 - 5"
2200 PRINT
2210 GET R$:IF R$ = "" THEN 2210
2220 R = RND(-TI)
2230 R = VAL(R$):IF R < 1 OR R > 5 THEN 2190
2240 ON R GOTO 300,400,500,700,950
2500 POKE 54277,128:POKE 54278,128
2510 POKE 54273,48:POKE 54272,127
2520 POKE 54276,17
2530 FOR E = 1 TO 250:NEXT E
2540 POKE 54277,0:POKE 54278,0
```

```
2550 POKE 54273,0:POKE 54272,0
2560 POKE 54276,0:RETURN
3010 DATA .----.,-.--.-,X,X,X
3020 DATA --..--,-....-,.-.-.-
3030 DATA -..-.,-----,.----
3040 DATA ..---,...--,....-
3050 DATA .....,-....,--...
3060 DATA ---..,----.,---...
3070 DATA -.-.-.,X,X,X,..--..,X
3080 DATA .-,-...,-.-.,-..,.
3090 DATA ..-.,--.,....,..,.---
3100 DATA -.-,.-..,--,-.,---
3110 DATA .--.,--.-,.-.,...,-
3120 DATA ..-,...-,.--,-..-
3130 DATA -.--,--..

READY.
```

EASY CHANGES

1. To change the speed of the codes or the pitch at which they are sounded, change line 2030. T is the speed (time factor) and HI and LO are the note. Change T to 1 to make the speed about 19 or 20 words per minute, the top speed. Change T to 10 for about eight words per minute. Changing the speed only changes the length of time between letters and words, not the speed of dots and dashes or the short time interval between them. Currently the note is low C. Consult the Commodore 64 User's Guide (Appendix M) for other note values. For example, to get a higher pitched sound (Middle C) with the fastest speed, make this change:

   2030 HI=17:LO=37:T=1
2. Change the number of characters in the multi-character quiz (option 4) by changing the value of N in line 2080.
3. Many experts stress that code is a language of *sound*, not sight, and should be learned that way. If you like, you can eliminate the displaying of dots and dashes on the screen by deleting line 170 and changing these lines:

   210 RETURN
   280 FOR J=1 TO 90:NEXT
4. Eliminate the sounding of "HAM CODE" at the start of the program by deleting line 2070.

5. To drill on only alphabetic characters during options three and four, make this change:

   900 R = INT(RND(1)*26) - 1

6 . A short delay is built into the program at several points. To lengthen it, replace the 800 in line 920 with 2000. To eliminate the delay, replace the 800 with 1.

MAIN ROUTINES

130- 220	Subroutine to sound and display character R$.
250- 290	Subroutine to sound and display phrase P$.
300- 360	Teaches characters by echoing keys until CLR/HOME is pressed.
400- 450	Teaches phrases by echoing entries until END is entered.
500- 620	Quizzes individual characters until CLR/HOME is pressed.
700- 800	Quizzes random N character phrases until END is entered.
900- 910	Subroutine to pick random character R$.
920	Delay subroutine.
2000-2030	Initializes variables. Stores codes in C$ array.
2050-2080	Displays and sounds title. Initializes more variables.
2100-2240	Displays options. Gets response. Initializes RND. Goes to option entered.
2500-2560	Sounds error code when a mistake is made. The sound is F# in the 5th octave.
3010-3130	DATA statements with codes for ASCII 39 (apostrophe) through 90 (Z). X value is illegal code. A through Z are in 3080 through 3130.

MAIN VARIABLES

W	Work variable and subscript.
R$	Character to be sounded; work character.
T$	Work string.
C$	Array of code strings.
J,K	Loop and work variables.
W$	Element (dot or dash) of code to be sounded.
D	Duration of sound to be made (dot = 1, dash = 3). Also loop variable.

T	Time factor to alter pauses between characters and words.
P$	Phrase of characters to be sounded.
L$	Last phrase entered.
N	Number of characters in multi-character quiz.
R	Random number for character selection; work variable.
HI,LO	Values of note to be sounded.

SUGGESTED PROJECTS

1. Add another option to randomly quiz the operator on a series of common words and/or phrases that have been stored in DATA statements.
2. Program some "intelligence" into the learning phase of the program. Have the program teach 3 or 4 common letters until the operator has mastered them, then begin teaching 3 or 4 more, etc.
3. Determine how to interface your Commodore 64 with amateur radio equipment so you can use the program to actually send code automatically under program control.
4. Now try the reverse of Project 3 – have the computer figure out how to decode a transmission that has been received over the radio, converting it into text. This will almost undoubtedly require some or all of the program to be in assembler language in order to run fast enough to handle this job in real time.

METRIC

PURPOSE

In case you don't realize it, we live in a metric world. The United States is one of the last holdouts, but that is changing rapidly. So if you're still inching along or watching those pounds, it's time to convert.

METRIC is an instructional program designed to familiarize you with the metric system. It operates in a quiz format; the program randomly forms questions from its data resources. You are then asked to compare two quantities—one in our old English units and one in the corresponding metric units. When you are wrong, the exact conversion and the rule governing it are given.

The two quantities to compare are usually within 50% of each other. Thus, you are constantly comparing an "English" quantity and a metric one which are in the same ball park. This has the effect of providing you some insight by sheer familiarity with the questions.

HOW TO USE IT

The program first asks how many questions you would like to do for the session. Any value of one or higher is acceptable.

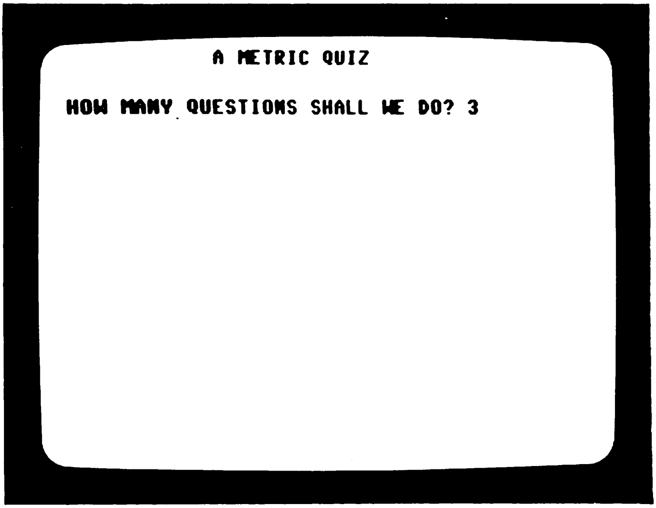
The sample run shows how each question is formulated. A quantity in English units is compared with one in metric units. Either one may appear first in the question. Each quantity will have an integral value. The relating word ("longer," "hotter," "heavier," etc.) indicates what type of quantities are being compared.

There are three possible replies to each question. Pressing **Y** or **N** means that you think the answer is yes or no, respectively. Pressing any other key indicates that you have no idea as to the correct answer.

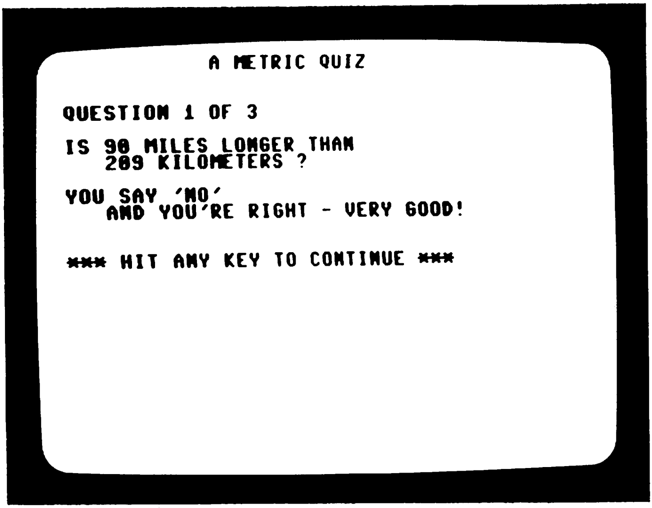
If you answer the question correctly, you will be duly congratulated and the program will proceed to the next question. A wrong answer or a response of "no idea," however, will generate some diagnostic information. The first value used in the question will be shown converted to its exact equivalent in the corresponding units. Also, the rule governing the situation will be displayed. At the end of any question, the program will request that you hit any key to proceed to the next question.

The program will continue generating the requested number of questions. Before ending, it will show you how many correct answers you gave and your percentage correct.

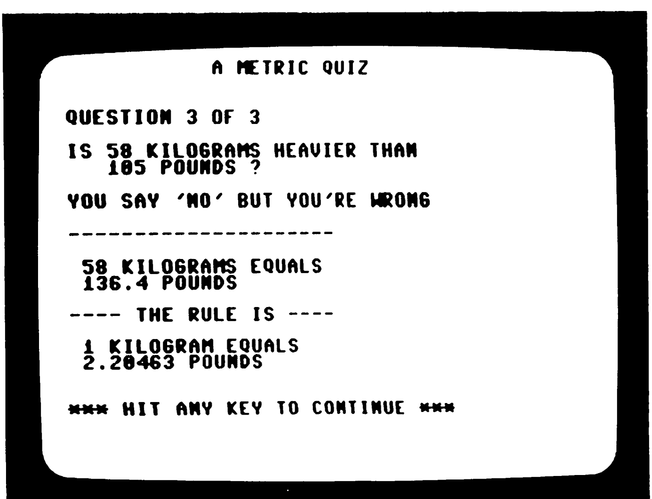
SAMPLE RUN



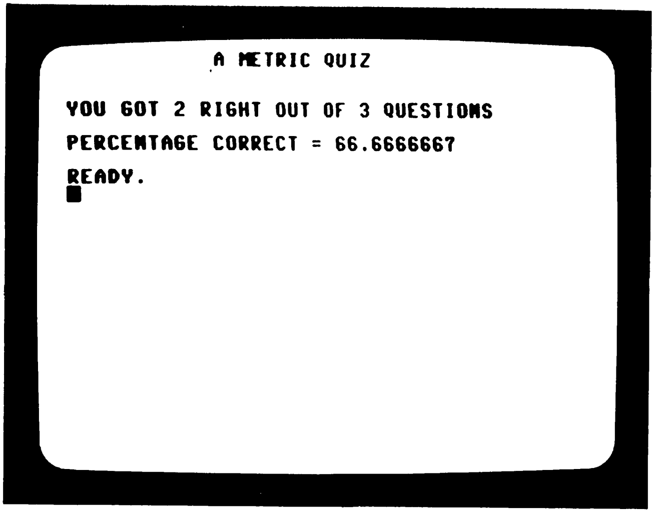
The operator requests a three question quiz.



The first question is correctly answered "no". The program waits for a key to be pressed before continuing the quiz.



Later, the third question is incorrectly answered "no". The correct conversion and governing rule are then displayed.



The program shows the number and percentage of correctly answered questions.

PROGRAM LISTING

```
READY.


100 REM: METRIC
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
120 CLR
150 DIM ES$(30),MS$(30),R$(30)
155 DIM C(30),EP$(30),MP$(30)
160 Q=RND(-TI):B$=" "
200 GOSUB 400:GOSUB 450
210 INPUT"HOW MANY QUESTIONS SHALL WE DO";NQ
215 NQ=INT(NQ):IF NQ<1 THEN 210
220 FOR J=1 TO NQ:GOSUB 600:GOSUB 900:NEXT
230 GOSUB 450:PRINT"YOU GOT";NR;"RIGHT OUT OF";
235 PRINT NQ;"QUESTIONS":PRINT
240 P=100*NR/NQ:PRINT"PERCENTAGE CORRECT =";P
250 END
400 RESTORE:ND=0
410 ND=ND+1:READ ES$(ND),MS$(ND),R$(ND)
415 READ C(ND),EP$(ND),MP$(ND)
420 IF ES$(ND)<>"XXX" THEN 410
```

```
430 ND=ND-1:RETURN
450 PRINT CHR$(147);TAB(11);"A METRIC QUIZ"
455 PRINT:PRINT:RETURN
600 N=INT(ND*RND(1))+1
610 F=0:IF RND(1)>0.5 THEN F=1
620 V1=INT(RND(1)*99)+2:V3=V1*C(N)
625 IF F=1 THEN V3=V1/C(N)
630 IF N=1 THEN V3=(V1-32)/1.8
635 IF F=1 THEN V3=(V1*1.8)+32
640 V2=V3*(0.5+RND(1)):V2=INT(V2+0.5)
645 T=0:IF V2<V3 THEN T=1
650 GOSUB 450:PRINT"QUESTION";J;"OF";NQ:PRINT
660 IF F=0 THEN PRINT"IS";V1;EP$(N);B$;R$(N);
665 IF F=1 THEN PRINT"IS";V1;MP$(N);B$;R$(N);
667 PRINT " THAN"
670 IF F=0 THEN PRINT B$;B$;V2;MP$(N);
675 IF F=1 THEN PRINT B$;B$;V2;EP$(N);
677 PRINT " ?"
680 GET Q$:IF Q$="" THEN 680
700 IF Q$="Y" THEN PRINT:PRINT"YOU SAY 'YES'";
705 IF Q$="Y" THEN R=1:GOTO 730
710 IF Q$="N" THEN PRINT:PRINT"YOU SAY 'NO'";
715 IF Q$="N" THEN R=0:GOTO 730
720 PRINT:PRINT"YOU HAVE NO IDEA":R=2
730 X=T-R:IF R=2 THEN GOSUB 800:GOTO 760
740 IF X=0 THEN PRINT
742 IF X=0 THEN PRINT "   AND YOU'RE RIGHT -";
744 IF X=0 THEN PRINT " VERY GOOD!":NR=NR+1
746 IF X=0 THEN GOTO 760
750 PRINT" BUT YOU'RE WRONG":GOSUB 800
760 RETURN
800 PRINT:PRINT"--------------------":PRINT
810 IF F=0 THEN PRINT V1;EP$(N);
815 IF F=0 THEN PRINT " EQUALS":PRINT V3;MP$(N)
820 IF F=1 THEN PRINT V1;MP$(N);
825 IF F=1 THEN PRINT " EQUALS":PRINT V3;EP$(N)
830 PRINT:PRINT"---- THE RULE IS ----":PRINT
840 IF N=1 AND F=0 THEN GOSUB 880:RETURN
850 IF N=1 AND F=1 THEN GOSUB 885:RETURN
860 IF F=0 THEN GOSUB 890:RETURN
870 Q=INT(1.E5/C(N))/1.E5:PRINT" 1 ";MS$(N);
875 PRINT" EQUALS":PRINT Q;EP$(N):RETURN
880 PRINT" DEG.C = (DEG.F - 32)/1.8":RETURN
885 PRINT" DEG.F = (DEG.C * 1.8) + 32":RETURN
890 PRINT" 1 ";ES$(N);" EQUALS"
895 PRINT C(N);MP$(N):RETURN
900 PRINT:PRINT
905 PRINT"*** HIT ANY KEY TO CONTINUE ***"
910 GET Q$:IF Q$="" THEN 910
920 RETURN
1000 DATA DEGREE FAHRENHEIT,DEGREE CENTIGRADE
1005 DATA HOTTER,0.5
```

```
1010 DATA DEGREES FAHRENHEIT,DEGREES CENTIGRADE
1020 DATA MILE PER HOUR,KILOMETER PER HOUR
1025 DATA FASTER,1.60935
1030 DATA MILES PER HOUR,KILOMETERS PER HOUR
1040 DATA FOOT,METER,LONGER,0.3048
1050 DATA FEET,METERS
1060 DATA MILE,KILOMETER,LONGER,1.60935
1070 DATA MILES,KILOMETERS
1080 DATA INCH,CENTIMETER,LONGER,2.54
1090 DATA INCHES,CENTIMETERS
1100 DATA GALLON,LITRE,MORE,3.78533
1110 DATA GALLONS,LITRES
1120 DATA POUND,KILOGRAM,HEAVIER,0.45359
1130 DATA POUNDS,KILOGRAMS
1999 DATA XXX,XXX,XXX,0,XXX,XXX

READY.
```

EASY CHANGES

1. To have the program always ask a fixed number of questions, change line 210 to set NQ to the desired value. For example:

   210 NQ=10

   will cause the program to do 10 questions.
2. There are currently seven conversions built into the program:

N	*Type*	*English Unit*	*Metric Unit*
1	temperature	degrees F.	degrees C.
2	speed	miles/hour	kilometers/hour
3	length	feet	meters
4	length	miles	kilometers
5	length	inches	centimeters
6	volume	gallons	litres
7	weight	pounds	kilograms

   If you wish to be quizzed on only one type of question, set N to this value by changing line 600. Thus,

   600 N=4

   will cause the program to only produce questions comparing miles and kilometers. To add additional data to the program, see the first "Suggested Project."
3. You can easily have the questions posed in one "direction" only. To go only from English to metric units change line 610 to

   610 F=0

while to go from metric to English units use

```
610 F=1
```

4. You might want the converted value and governing rule to be displayed even when the correct answer is given. This is accomplished by changing lines 740 through 746 to read as follows:

```
740 IF X=0, THEN PRINT:PRINT" AND YOU'RE
                         RIGHT-VERY GOOD!"
746 IF X=0 THEN NR=NR+1: GOSUB 800: GOTO 760
```

MAIN ROUTINES

150- 160	Dimensions and initializes variables.
200- 250	Mainline routine, drives other routines.
400- 430	Reads and initializes data.
450- 455	Displays header.
600- 760	Forms and asks questions. Processes user's reply.
800- 895	Displays exact conversion and governing rule.
900- 920	Waits for user to hit any key.
1000-1999	Data statements.

MAIN VARIABLES

ND	Number of conversions in the data.
ES$,EP$	String arrays of English units' names (singular, plural).
MS$,MP$	String arrays of metric units' names (singular, plural).
R$	String array of the relation descriptors.
C	Array of the conversion factors.
Q	Work variable.
B$	String constant of one blank character.
J	Current question number.
NR	Number of questions answered right.
P	Percentage answered right.
NQ	Number of questions in session.
N	Index number of current question in the data list.
F	Flag on question "direction" (0 = English to metric; 1 = metric to English).
V1,V2	Numeric values on left, right sides of the question.
V3	The correct value of the right hand side.

T	Flag on the question's correct answer (1 = true; 0 = false).
Q$	User reply string.
R	User reply flag (0 = no; 1 = yes; 2 = no idea).
X	User's result (0 if correct answer was given).

SUGGESTED PROJECTS

1. Each built-in conversion requires six elements of data in this order:

Element	*Data Description*
1	English unit (singular)
2	Metric unit (singular)
3	Relation descriptor (e.g., "hotter," "faster," etc.)
4	Conversion factor (from English to metric)
5	English unit (plural)
6	Metric unit (plural)

   Each of these elements, except the fourth, is a string. The data statements in the listing should make clear how the information is to be provided. You can add new data to the program with appropriate data statements in this format. New data should be added after the current data, i.e. just before line 1999. Line 1999 is a special data statement to trigger the end of all data to the program. The program is dimensioned up to thirty entries while only seven are currently used. (Note: this format allows only conversions where one unit is a direct multiple of the other. Temperature, which does not fit this rule, is handled as a special case throughout the program.)
2. Convert the program to handle units conversion questions of any type.
3. Keep track of the questions asked and which ones were missed. Then do not ask the same questions too soon if they have been answered correctly. However, do re-ask those questions missed for additional practice.

NUMBERS

PURPOSE

This is an educational program for pre-school children. After a few weeks of watching Sesame Street on television, most three and four year old children will learn how to count from one to ten. The NUMBERS program allows these children to practice their numbers and have fun at the same time.

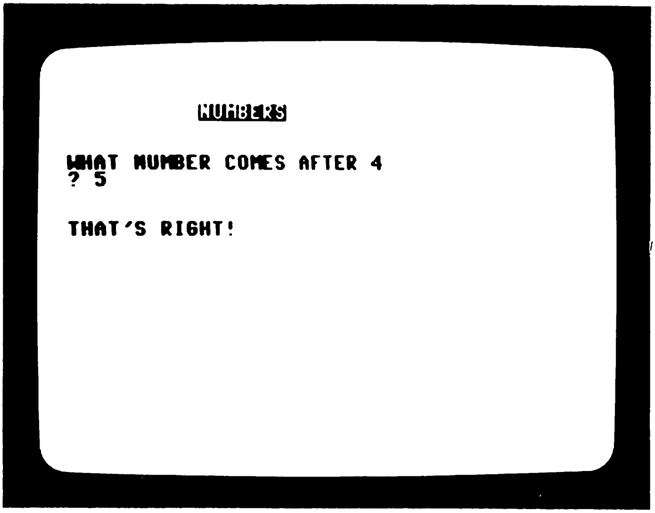
HOW TO USE IT

We know a child who learned how to type LOAD and RUN to get this program started before she turned three, but you'll probably have to help your child with this for a while. The program asks the question, "WHAT NUMBER COMES AFTER n?", where n is a number from one to eight. Even if the child can't read yet, he or she will soon learn to look for the number at the end of the line. The child should respond with the appropriate number, and then press the **RETURN** key.

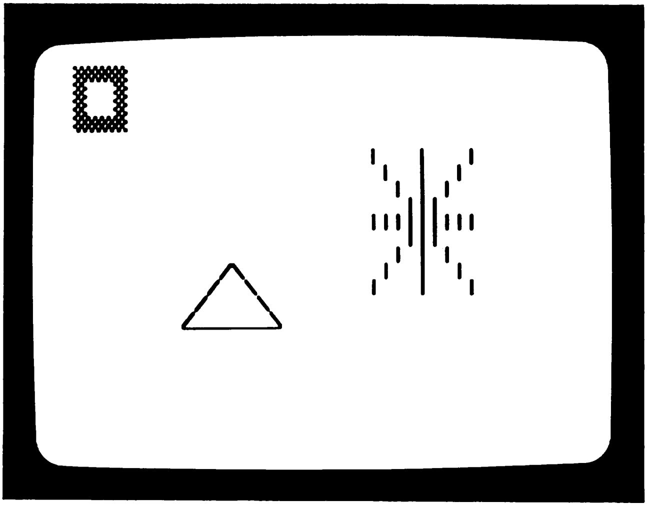
If the answer is correct, the program displays the message "THAT'S RIGHT!", pauses for a couple of seconds, and then clears the screen and displays three geometric shapes. In the upper left of the screen a square is drawn. In the lower center, a triangle is drawn. Then an asterisk (or a snowflake, perhaps?) is drawn in the upper right portion of the screen. After a few seconds delay, the program clears the screen and asks another question. The same number is never asked twice in a row. The size of the three figures is chosen at random each time. If the child provides the wrong answer, a message indicates the error and the same question is asked again.

The program keeps on going until you hit the **RUN/STOP** key. Remember that most children have a pretty short attention span, so please do not force your child to continue after his or her interest diminishes. Keep each session short and fun. This way, it will always be a treat to "play" with the computer.

SAMPLE RUN



The program asks what number comes after 4, and waits for a response. The operator says 5, and the program acknowledges that the answer is correct.



Because of the correct response, the program draws three geometric figures.

PROGRAM LISTING

```
READY.

100 REM: NUMBERS
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 M=9
130 S=1024:E=10:TS=12:CC=55296
140 R=RND(-TI):PRINT CHR$(147)
150 PRINT:PRINT
160 PRINT TAB(10);CHR$(18);"NUMBERS"
170 R=INT(M*RND(1))+1:IF R=P THEN 170
180 PRINT:PRINT
190 PRINT"WHAT NUMBER COMES AFTER";R
200 INPUT R$
210 PRINT:PRINT
220 IF VAL(R$)=R+1 THEN 300
230 PRINT"NO, THAT'S NOT IT.  TRY AGAIN."
240 GOTO 180
```

```
300 PRINT"THAT'S RIGHT!"
310 FOR X=1 TO 1000:NEXT
320 P=R:C=102:POKE 53281,1:PRINT CHR$(147);
325 POKE 53280,1:POKE 53281,0
330 E=INT(8*RND(1))+3
335 C1 = INT(RND(1)*14)+1
400 Y=1:FOR X=1 TO E:GOSUB 900:NEXT
405 C1 = INT(RND(1)*14)+1
410 X=E:FOR Y=1 TO E:GOSUB 900:NEXT
415 C1 = INT(RND(1)*14)+1
420 Y=E:FOR X=E TO 1 STEP -1:GOSUB 900:NEXT
425 C1 = INT(RND(1)*14)+1
430 X=1:FOR Y=E TO 1 STEP -1:GOSUB 900:NEXT
435 C1 = INT(RND(1)*14)+1
450 C=77:FOR J=1 TO E
460 Y=TS+J:X=Y:GOSUB 900:NEXT
465 C1 = INT(RND(1)*14)+1
470 C=78:FOR J=1 TO E
480 Y=TS+J:X=TS-J+1:GOSUB 900:NEXT
490 C=99:Y=TS+E+1:FOR X=TS-E+1 TO TS+E
495 C1 = INT(RND(1)*14)+1
500 GOSUB 900:NEXT
520 A=28:B=10:C=INT(7*RND(1))+86
530 FOR J=1 TO E
540 X=A+J:Y=B+J:GOSUB 900
550 Y=B-J:GOSUB 900
560 Y=B:GOSUB 900
570 X=A:GOSUB 900
580 Y=B+J:GOSUB 900
590 Y=B-J:GOSUB 900
600 X=A-J:GOSUB 900
610 Y=B:GOSUB 900
620 Y=B+J:GOSUB 900
630 NEXT
800 FOR J=1 TO 3000:NEXT J
810 POKE 53281,1:PRINT CHR$(147);
815 POKE 53280,14:POKE 53281,6
820 GOTO 170
900 POKE S+40*Y+X,C
905 POKE CO+40*Y+X,C1
910 RETURN

READY.
```

EASY CHANGES

1. Change the range of numbers that the program asks by altering the value of M in line 120. For a beginner, use a value of 3 for M instead of 8. Later, increase the value of M to 5, and then 8.

2. Alter the delay after "THAT'S RIGHT!" is displayed by altering the value of 1000 in statement 310. Double it to double the time delay, etc. The same can be done with the 2000 in line 800 to alter the delay after the figures are drawn.
3. To avoid randomness in the size of the figures that are drawn, replace line 330 with

   330 E=10

   Instead of 10, you can use any integer from 2 to 11.
4. To slowly increase the size of the figures from small to large as correct answers are given (and the reverse for incorrect answers), do the following:
   a. Replace the 10 in line 130 with a 2.
   b. Insert this line:

      225 E=E−3: IF E<2 THEN E=2

   c. Replace line 330 with the following:

      330 E=E+2: IF E>11 THEN E =11

MAIN ROUTINES

120-160	Initializes variables. Clears screen.
170	Picks random integer from 1 to M.
180-240	Asks question. Gets answer. Determines if right or wrong.
310	Delays about 2 seconds.
320-430	Draws a square.
450-500	Draws a triangle.
520-630	Draws an asterisk.
800	Delays about 4 seconds.
810-820	Clears screen. Goes back to ask next question.
900-910	Subroutine to POKE graphics character C to X,Y coordinate on screen. POKEs color to corresponding location on color map.

MAIN VARIABLES

M	Maximum number that will be asked.
E	Edge length of geometric figures.
R	Random integer in range from 1 to M.
P	Previous number that was asked.
R$	Reply given by operator.

X,Y	Coordinates in CRT display.
TS	Triangle's starting location (top).
A,B	X,Y coordinate values.
J	Subscript variable.
C	Graphics character to be POKEd.
CO	Starting location of color map.
C1	Color for current graphics character.
S	Starting address of CRT screen.

SUGGESTED PROJECTS

1. Modify the program to ask the next letter of the alphabet. Use the ASC and CHR$ functions in picking a random letter from A to Y, and to check whether the response is correct or not.
2. Ask each number from 1 to M once (in a random sequence). At the end of the sequence, repeat those that were missed.
3. Add different shapes to the graphics display that is done after a correct answer. Try an octagon, a diamond, and a rectangle. Or, combine this program with one of the graphics display programs.

TACHIST

PURPOSE

This program turns your computer into a tachistoscope (tah-KISS-tah-scope). A tachistoscope is used in reading classes to improve reading habits and, as a result, improve reading speed. The program displays a word or phrase on the screen for a fraction of a second, then asks you what it was. With a little practice, you will find that you can read phrases that are displayed for shorter and shorter time periods.

HOW TO USE IT

The program starts off by displaying a brief introduction and waiting for you to press any key (except the **RUN/STOP** key or **SHIFT** keys, of course). After you press a key, the screen is blanked out except for two horizontal dash lines in the upper left-hand corner. After two and a half seconds, the phrase is flashed on the screen between the two lines. Then the screen is blanked again, and you are asked what the phrase was.

If you respond correctly, the next phrase is displayed for a shorter time period (half as long). If you respond incorrectly, the program shows you the correct phrase, and the next phrase is displayed for a longer period of time (twice as long).

The fastest the computer can display a phrase and erase it is about .02 seconds (one-fiftieth). See if you can reach the top speed and still continue to read the phrases correctly.

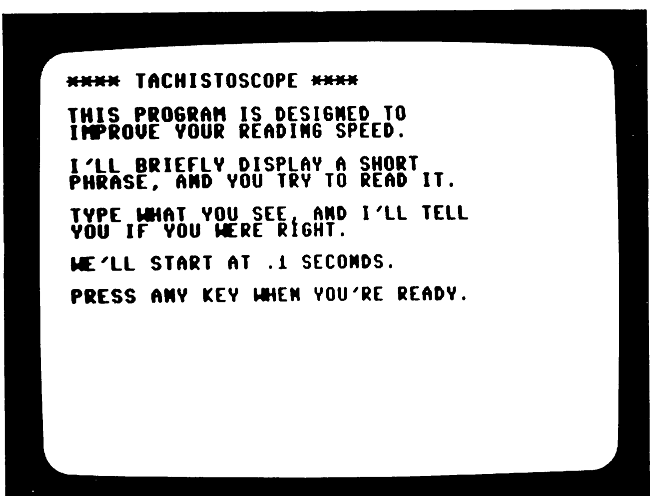
A great deal of research has been done to determine how people read and what they should do to read both faster and with

better comprehension. We will not try to explain it all (see the bibliography), but a couple of things are worth mentioning.

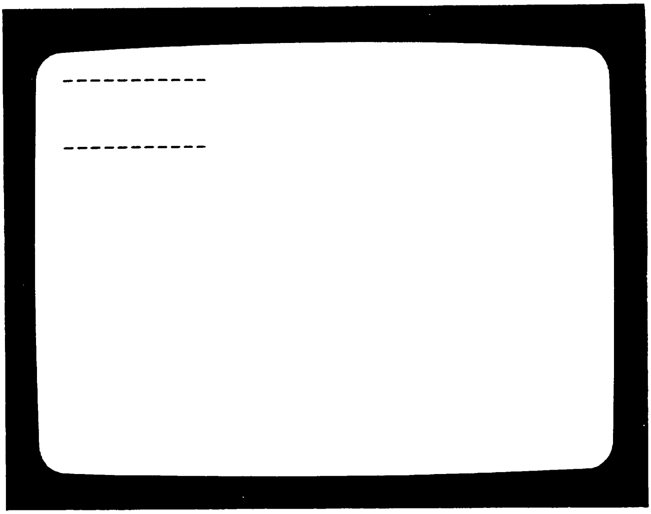
To read fast, you should not read one word at a time. Instead, you should learn to quickly read an entire phrase at once. By looking at a point in the center of the phrase (and slightly above it), your eyes can see the whole phrase *without* the necessity of scanning it from left to right, word by word. Because the tachistoscope flashes an entire phrase on the screen at once, it forces you to look at a single point and absorb the whole phrase, rather than scanning left to right, word by word.

If you can incorporate this technique into your reading and increase the width of the phrases you absorb, your reading speed can increase dramatically.

SAMPLE RUN



The program displays an introduction, then waits.



The program clears the screen and displays two parallel lines that remain in the upper left corner of the screen for a couple of seconds.



The program flashes a short phrase (chosen at random) between the two lines for a fraction of a second, then clears the screen.



The program asks what the phrase was. The operator responds correctly. The program acknowledges the correct response, and indicates that the next phrase will be shown for half as long.

PROGRAM LISTING

```
READY.

100 REM: TACHIST
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 T=.1
130 J=T*60:B=147
140 L=50
150 DIM T$(L)
160 C=0
170 READ R$
180 IF R$="XXX" THEN 250
190 C=C+1
200 IF C>L THEN PRINT"TOO MANY DATA STATEMENTS"
205 IF C>L THEN END
210 T$(C)=R$
220 GOTO 170
```

```
250 R=RND(-TI)
260 PRINT CHR$(B)
270 PRINT"**** TACHISTOSCOPE ****"
280 PRINT
290 PRINT"THIS PROGRAM IS DESIGNED TO"
300 PRINT"IMPROVE YOUR READING SPEED."
310 PRINT
320 PRINT"I'LL BRIEFLY DISPLAY A SHORT"
330 PRINT"PHRASE, AND YOU TRY TO READ IT."
340 PRINT
350 PRINT"TYPE WHAT YOU SEE, AND I'LL TELL"
360 PRINT"YOU IF YOU WERE RIGHT."
370 PRINT
380 PRINT"WE'LL START AT";T;"SECONDS."
400 FOR K=1 TO 5:GET R$:NEXT:PRINT
410 PRINT"PRESS ANY KEY WHEN YOU'RE READY."
420 GET R$:IF R$="" THEN 420
430 R=INT(C*RND(1))+1
440 IF R=P1 OR R=P2 OR R=P3 THEN 430
450 IF R=P4 OR R=P5 THEN 430
460 PRINT CHR$(B):GOSUB 840
465 FOR K=1 TO 1500:NEXT K:IF J<2 THEN 800
470 PRINT:PRINT:PRINT T$(R)
480 S=TI
490 IF TI-S<J THEN 490
500 PRINT CHR$(B)
505 FOR K=1 TO 500:NEXT K
510 PRINT:PRINT:PRINT:PRINT
520 PRINT"WHAT WAS IT?"
530 INPUT R$
540 PRINT
550 IF R$<>T$(R) THEN 700
560 PRINT" THAT'S RIGHT!"
570 J=J-3
580 IF J<1.2 THEN J=1.2
590 PRINT
600 P1=P2:P2=P3:P3=P4:P4=P5:P5=R:PRINT
610 PRINT"THE NEXT ONE WILL BE DISPLAYED"
620 PRINT"FOR";J/60;"SECONDS."
630 PRINT
640 GOTO 400
700 PRINT"NO, THAT'S NOT IT.  IT WAS"
710 PRINT:PRINT"'";T$(R);"'"
720 J=J+3
730 IF INT(J/3)<>J/3 THEN J=3*INT(J/3)
740 GOTO 590
800 PRINT:PRINT:PRINT T$(R);CHR$(147)
810 GOTO 505
840 PRINT"----------":PRINT
850 PRINT:PRINT:PRINT"----------"
```

```
860 PRINT CHR$(19)
870 RETURN
910 DATA"AT THE TIME"
920 DATA"THE BROWN COW"
930 DATA"LOOK AT THAT"
940 DATA"IN THE HOUSE"
950 DATA"THIS IS MINE"
960 DATA"SHE SAID SO"
970 DATA"THE BABY CRIED"
980 DATA"TO THE STORE"
990 DATA"READING IS FUN"
1000 DATA"HE GOES FAST"
1010 DATA"IN ALL THINGS"
1020 DATA"GREEN GRASS"
1030 DATA"TWO BIRDS FLY"
1040 DATA"LATE LAST NIGHT"
1050 DATA"THEY ARE HOME"
1060 DATA"ON THE PHONE"
1070 DATA"THROUGH A DOOR"
1080 DATA"WE CAN TRY"
1090 DATA"MY FOOT HURTS"
1100 DATA"HAPPY NEW YEAR"
9999 DATA XXX

READY.
```

EASY CHANGES

1. Change the phrases that are displayed by changing the DATA statements that start at line 910. Add more and/or replace those shown with your own phrases or words. Line 140 must specify a number that is at least as large as the number of DATA statements. So, to allow for up to 100 DATA statements, change line 140 to say

   140 L=100

   Be sure to enter your DATA statements in the same form shown in the program listing. To begin with, you may want to start off with shorter phrases or single words. Later, try longer phrases. Do not alter line 9999, which has to be the last DATA statement. Be sure to have at least 6 DATA statements.
2. To change the length of time the first phrase is displayed, change the value of T in line 120. Use a multiple of .05 seconds or make it .02 (the fastest speed).

3. To cause all phrases to be displayed for the same length of time, remove lines 570 and 720.
4. If you want to change the waiting period before the phrase is flashed on the screen, change the 1500 in line 465. To make the delay five seconds, change it to 3000. To make it one second, change it to 600.
5. To put the program into a sort of flashcard mode, in which the phrases are flashed, but no replies are necessary, insert these three lines:

```
515 GOTO 710
715 GOTO 590
```

   This will cause each phrase to be flashed (all for the same length of time), and then displayed again so you can verify what it was.

MAIN ROUTINES

120- 150	Initializes variables.
160- 220	Reads DATA statements into T$ array.
260- 380	Displays introduction.
400- 420	Waits for operator to press a key.
430- 450	Picks random phrase from T$ array. Ensures no duplication from previous five phrases.
460- 465	Clears screen and displays horizontal lines.
470- 500	Displays phrase for appropriate length of time.
505- 530	Asks what the phrase was.
550	Determines if typed phrase matches the phrase displayed.
560- 640	Shortens time for next phrase if reply was correct. Saves subscript to avoid repetition. Goes back to wait for key to be pressed.
700- 740	Shows what phrase was. Lengthens time for next phrase. Ensures that time period is a multiple of .05 seconds.
800- 810	Special routine to display phrase for shortest time (.02 seconds).
840- 870	Subroutine to display horizontal dash lines.
910-9999	DATA statements with phrases to be displayed.

MAIN VARIABLES

S	Starting time of display of phrase (in intervals).
T	Time that phrase will be displayed.
J	Loop variable and time interval counter.
L	Limit of number of phrases.
T$	Array of phrases (read into from DATA statements).
C	Count of number of phrases actually read.
R$	Temporary string variable. Also, reply of operator.
R	Work variable. Also, subscript of phrase to be displayed.
P1,P2, P3,P4,P5	Subscripts of the five previous phrases.
K	Temporary work variable.
B	ASCII number for clear screen character.

SUGGESTED PROJECTS

1. Instead of picking phrases at random, go through the list once sequentially.
2. Instead of only verifying that the current phrase does not duplicate any of the previous four phrases, modify the program to avoid duplication of the previous ten or more. Changes will be needed to lines 440, 450, and 600.
3. Keep score of the number of correct and incorrect replies, and display the percentage each time. Alternatively, come up with a rating based on the percentage correct and the speed attained, possibly in conjunction with a difficulty factor for the phrases used.
4. Add the capability to the program to also have a mode in which it can display a two to seven digit number, chosen at random. Have the operator try several of the numbers first (maybe five-digit ones) before trying the phrases. The phrases will seem easy after doing the numbers.

VOCAB

PURPOSE

Did you ever find yourself at a loss for words? Well, this vocabulary quiz can be used in a self-teaching environment or as reinforcement for classroom instruction to improve your ability to remember the jargon of any subject. It allows you to drill at your own pace, without the worry of ridicule from other students or judgment by an instructor. When you make mistakes, only the computer knows, and it's not telling anyone except you. Modifying the program to substitute a different vocabulary list is very simple, so you can accumulate many different versions of this program, each with a different set of words.

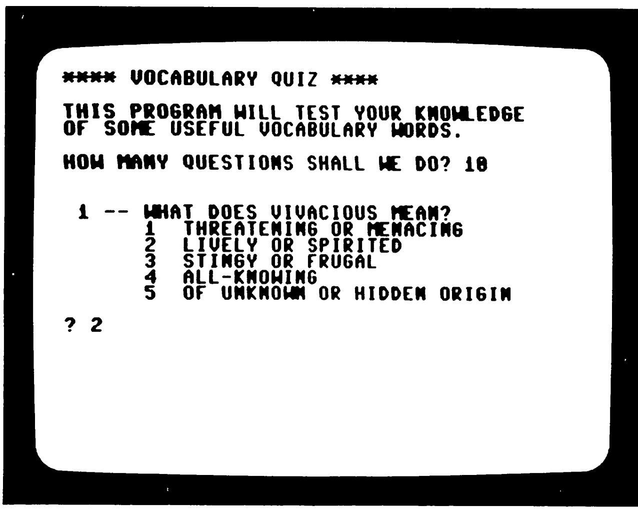
HOW TO USE IT

This program is pretty much self-explanatory from the sample run. After you enter "RUN," it asks you how many questions you would like. If you respond with less than five, it will still do five. Otherwise, it will do the number you enter.

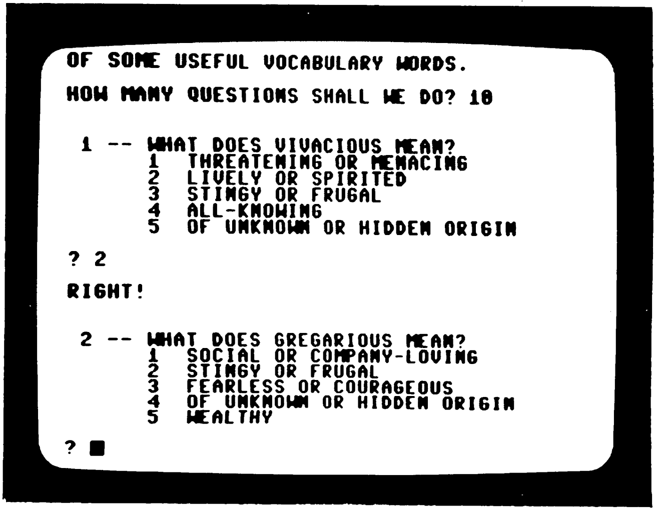
Next, you get a series of multiple choice questions. Each question is formatted in one of two ways—either you are given a word and asked to select from a list of definitions, or you are given a definition and asked to select from a list of words. The format is chosen at random. You respond with the number of the choice you think is correct. If you are right, you are told so. If not, you are shown the correct answer. From the second answer on, you are shown a status report of the number correct out of the number attempted so far.

Finally, after the last question, you are shown the percentage you got correct, along with a comment on your performance. Then you have the option of going back for another round of questions or stopping.

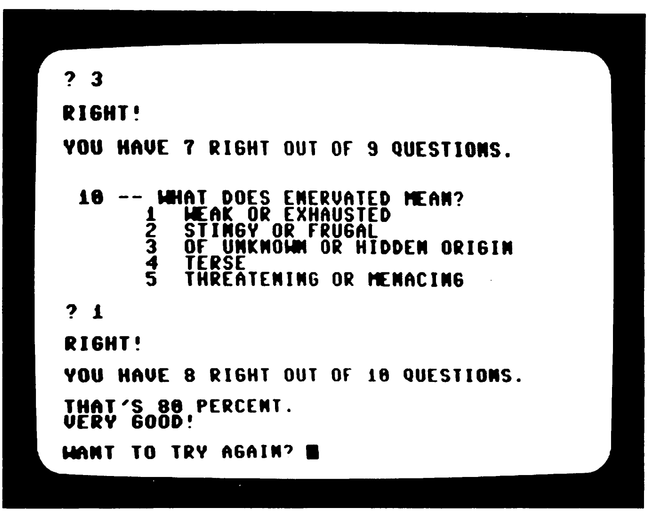
SAMPLE RUN



The program displays an introduction and asks the first question. The operator selects choice 2.



The program responds that the first answer was correct, and asks the next question.



At the end of ten questions, the program gives a final score and asks about trying again.

PROGRAM LISTING

```
READY.

100 REM: VOCAB
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
300 GOSUB 1000
400 GOSUB 2000
500 GOSUB 3000
600 GOSUB 4000
700 GOSUB 5000
800 GOSUB 6000
900 IF E=0 THEN 500
910 GOTO 300
990 REM
1000 IF E<>0 THEN 1060
1010 PRINT CHR$(147)
1020 PRINT"**** VOCABULARY QUIZ ****"
1030 PRINT
1040 PRINT"THIS PROGRAM WILL TEST YOUR KNOWLEDGE"
1050 PRINT"OF SOME USEFUL VOCABULARY WORDS."
1060 PRINT
1110 INPUT"HOW MANY QUESTIONS SHALL WE DO";L
1120 L=INT(L)
1130 IF L>4 THEN 1145
1135 PRINT"THAT'S NOT ENOUGH.  LET'S DO 5."
1140 L=5
1145 IF E<>0 THEN 1200
1150 PRINT
1160 R=RND(-TI)
1200 RETURN
2000 IF E<>0 THEN 2200
2010 C=5
2020 D=26
2030 DIM D$(D),E$(D)
2040 DIM P(C)
2050 J=1
2060 READ D$(J)
2070 IF D$(J)="XXX" THEN 2140
2090 READ E$(J)
2100 J=J+1
2110 IF J<=D THEN 2060
2120 PRINT"TOO MANY DATA STATEMENTS."
2130 PRINT"ONLY FIRST";D;"ARE USED."
2140 D=J-1
2200 Q=1
2210 E=0
```

```
2220 Q1=0
2300 RETURN
3000 FOR J=1 TO C
3010 P(J)=0
3020 NEXT J
3030 FOR J=1 TO C
3040 P=INT(D*RND(1))+1
3045 IF P=P1 OR P=P2 OR P=P3 THEN 3040
3050 FOR K=1 TO J
3060 IF P(K)=P THEN 3040
3070 NEXT K
3080 P(J)=P
3090 NEXT J
3110 A=INT(C*RND(1))+1
3200 RETURN
4000 PRINT
4010 M=RND(1)
4020 IF M>.5 THEN 4100
4030 PRINT Q;"-- WHAT WORD MEANS "
4035 PRINT TAB(6);E$(P(A));"?"
4040 FOR J=1 TO C
4050 PRINT TAB(5);J;" ";D$(P(J))
4060 NEXT J
4070 GOTO 4200
4100 PRINT Q;"-- WHAT DOES ";D$(P(A));" MEAN?"
4110 FOR J=1 TO C
4120 PRINT TAB(5);J;" ";E$(P(J))
4130 NEXT J
4200 PRINT
4210 RETURN
5000 INPUT R
5010 R=INT(R)
5020 IF R>=1 AND R<=C THEN 5040
5030 PRINT"I NEED A NUMBER FROM 1 TO";C
5035 GOTO 5000
5040 PRINT
5050 IF R=A THEN 5100
5060 PRINT"NO, THE ANSWER IS NUMBER";A
5070 GOTO 5200
5100 PRINT"RIGHT!"
5110 Q1=Q1+1
5200 PRINT
5210 IF Q=1 THEN 5300
5220 PRINT"YOU HAVE";Q1;
5225 PRINT"RIGHT OUT OF";Q;"QUESTIONS."
5230 PRINT
5300 P3=P2
5310 P2=P1
5320 P1=P(A)
5330 RETURN
```

```
6000 Q=Q+1
6010 IF Q<=L THEN RETURN
6020 E=1
6030 Q=INT(Q1*100/(Q-1))
6040 IF Q>0 THEN 6070
6050 PRINT"WELL, THAT'S A 'PERFECT' SCORE..."
6060 GOTO 6200
6070 PRINT"THAT'S";Q;"PERCENT."
6080 IF Q>25 THEN 6110
6090 PRINT"CONGRATULATIONS ON AVOIDING A SHUTOUT."
6100 GOTO 6200
6110 IF Q>50 THEN 6140
6120 PRINT"YOU CAN USE SOME MORE PRACTICE."
6130 GOTO 6200
6140 IF Q>75 THEN 6170
6150 PRINT"NOT BAD, BUT ROOM FOR IMPROVEMENT."
6160 GOTO 6200
6170 PRINT"VERY GOOD!"
6180 IF Q>95 THEN PRINT"YOU'RE ALMOST AS";
6185 IF Q>95 THEN PRINT" SMART AS I AM!"
6200 PRINT
6210 INPUT"WANT TO TRY AGAIN";R$
6220 IF LEFT$(R$,1)<>"N" THEN 6230
6225 PRINT:PRINT"CHECK YOU LATER.":PRINT
6228 END
6230 IF LEFT$(R$,1)="Y" THEN 6250
6240 GOTO 6210
6250 RETURN
7000 REM: ON LINE 2020, D MUST BE AT
7002 REM: LEAST ONE GREATER THAN THE
7004 REM: NUMBER OF DIFFERENT WORDS.
7010 DATA ANONYMOUS,"OF UNKNOWN OR HIDDEN ORIGIN"
7020 DATA OMINOUS,"THREATENING OR MENACING"
7030 DATA AFFLUENT,WEALTHY
7040 DATA APATHETIC,"INDIFFERENT OR UNINTERESTED"
7050 DATA LACONIC,TERSE
7060 DATA INTREPID,"FEARLESS OR COURAGEOUS"
7070 DATA GREGARIOUS,"SOCIAL OR COMPANY-LOVING"
7080 DATA ENERVATED,"WEAK OR EXHAUSTED"
7090 DATA VENERABLE
7095 DATA "WORTHY OF RESPECT OR REVERENCE"
7100 DATA DISPARATE,"DIFFERENT AND DISTINCT"
7110 DATA VIVACIOUS,"LIVELY OR SPIRITED"
7120 DATA ASTUTE,"KEEN IN JUDGMENT"
7130 DATA URSINE,BEARLIKE
7140 DATA PARSIMONIOUS,"STINGY OR FRUGAL"
7150 DATA OMNISCIENT,"ALL-KNOWING"
7999 DATA XXX

READY.
```

EASY CHANGES

1. Add more DATA statements between lines 7010 and 7999, or replace them all with your own. Be careful not to use two or more words with very similar definitions; the program might select more than one of them as possible answers to the same question. Note that each DATA statement first has the vocabulary word, then a comma, and then the definition or synonym. Be sure there are no commas or colons in the definition (unless you enclose the definition in quotes). If you add more DATA statements, you have to increase the value of D in line 2020 to be at least one greater than the number of words. The number of DATA statements you can have depends on how long each one is and how much user memory your computer has. Be sure to leave statement 7999 as it is—it signals that there are no more DATA statements.
2. To get something other than five choices for each question, change the value of C in line 2010. You might want only three or four choices per question.
3. If you do not want the choice of how many questions, remove lines 1110 through 1130. Line 1140 establishes the number of questions. Change it to the number you would like.
4. To make the program pause longer after a wrong answer, insert:

```
5065 FOR J=1 TO 5000:NEXT
```

MAIN ROUTINES

300- 910	Mainline routine. Calls major subroutines.
1000-1200	Displays introduction. Initializes RND function. Displays number of questions to be asked.
2000-2300	Reads vocabulary words and definitions into arrays. Performs housekeeping.
3000-3200	Selects choices for answers and determines which will be the correct one.
4000-4210	Determines in which format the question will be asked. Asks it.
5000-5330	Accepts answer from operator. Determines if right or wrong. Keeps score. Saves subscripts of last three correct answers.

6000-6250	Gives final score. Asks about doing it again.
7010-7999	DATA statements with vocabulary words and definitions.

MAIN VARIABLES

E	Set to 1 to avoid repeating introduction after the first round.
L	Limit of number of questions to ask.
R	Work variable. Also used for operator's reply to each question.
C	Number of choices of answers given for each question.
D	At least one greater than number of DATA statements. Used to DIM arrays.
D$	Array of vocabulary words.
E$	Array of definitions.
P	Array for numbers of possible answers to each question.
J	Work variable (subscript for FOR-NEXT loops).
Q	Number of questions asked so far (later used to calculate percent correct).
Q1	Number of questions correct so far.
P	Work variable.
P1,P2,P3	Last three correct answers.
A	Subscript of correct answer in P array.
M	Work variable to decide which way to ask question.
R$	Yes or no reply about doing another round.

SUGGESTED PROJECTS

1. Modify lines 6030 through 6200 to display the final evaluation messages based on a finer breakdown of the percent correct. For example, show one message if 100 percent, another if 95 to 99, another if 90 to 94, etc.
2. Ask the operator's name in the introduction routine, and personalize some of the messages with his/her name.
3. Instead of just checking about the last three questions, be sure that the next question has not been asked in the last eight or ten questions. (Check lines 3045 and 5300 through 5320.)
4. Keep track of which questions the operator misses. Then, after going through the number of questions he/she requested, repeat those that were missed.

Section 3

Game Programs

INTRODUCTION TO GAME PROGRAMS

Almost everyone likes to play games. Computer games are a fun and entertaining use of your Commodore 64. Besides providing relaxation and recreation, they have some built-in practical bonuses. They often force you to think strategically, plan ahead, or at least be orderly in your thought processes. They are also a good way to help some friends over their possible "computer phobia." We present a collection of games to fit any game playing mood.

Maybe you desire a challenging all-skill game? Like chess or checkers, WARI involves no luck and considerable thinking. The computer will be your opponent, and a formidable one indeed.

Perhaps you're in the mood for a game with quick action and mounting excitement. GROAN is a fast-paced dice game involving mostly luck with a dash of skill (or intuition) thrown in. The Commodore 64 is ready to take you on anytime.

Two word games are included. In JOT, you and the computer each pick secret words and then try to home in on each other's selection. In ARGO, you are challenged to make words by unscrambling letters in a race against time.

Do you like solving puzzles? If so, try DECODE. The computer will choose a secret code and then challenge you to find it.

Graphic electronic arcade games are a prevalent landmark of our times. We include two such games. ROADRACE puts you behind the wheel of a high speed race car. You must steer ac-

curately to stay on course. OBSTACLE lets you and a friend compete in a game of cut and thrust. Each of you must avoid crossing the path laid by the other, and by yourself!

ARGO

PURPOSE

Argo is a word game that is both challenging and a lot of fun. The program displays 13 random letters, and your object is to try to score as many points as possible by creating words from them.

HOW TO USE IT

The program begins by displaying its name and asking you to press a key to start the game. After you press a key (other than **SHIFT** or **RUN/STOP**, of course), the program displays 13 letters in alphabetical order and starts its "timer." When the timer in the upper left corner reaches 5000, the game is over.

Your object is to create words that are at least three, but no more than seven letters long. You can enter as many as six words of each length, but only the first five will score points. This is to give you a chance to enter an extra word of some length in case you mis-typed a word or entered a word that is later disallowed.

Each word is entered by simply typing the letters of the word and pressing the **RETURN** key. As each letter of the word is typed, it is displayed at the top of the screen. When **RETURN** is pressed, the word is moved to the lower part of the screen, where a column of words is displayed for each length. The three letter words are at the left of the screen, and the seven letter words are at the right.

If a typing error is made before pressing **RETURN**, you can simply correct it as usual by using the "back arrow" key. If you

do not see the error until after you pressed **RETURN**, there is no way to erase the erroneous word.

The program displays an error message to the right of your word if you enter a duplicate word or if you try to enter a word that is not made up of the letters shown. Of course, you can only use a letter the number of times it is shown—to use a letter twice, there need to be two of them.

The program has no way of knowing whether or not you are entering legitimate words. It only checks that you are using the proper letters. It's up to you to determine if you want to allow slang, proper names, foreign words, etc. If you are going to compete with a friend, be sure you establish the ground rules first.

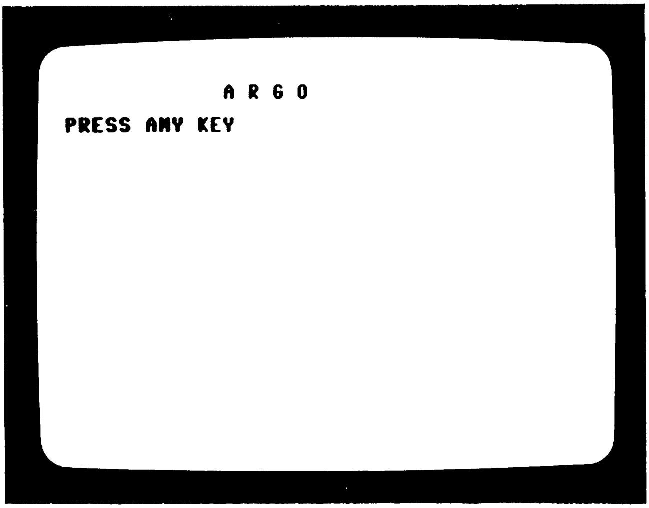
At the end of your time limit, the program displays the score and ends. Scoring is based on how many words you entered of each length. Each word counts the square of its word length in points. So, each three letter word counts nine points. Each four letter word is 16 points, a five letter word is 25 points, a six letter word is 36 points, and a seven letter word is 49 points. This means that the maximum possible score is

$$5 \times (9 + 16 + 25 + 36 + 49) = 675.$$

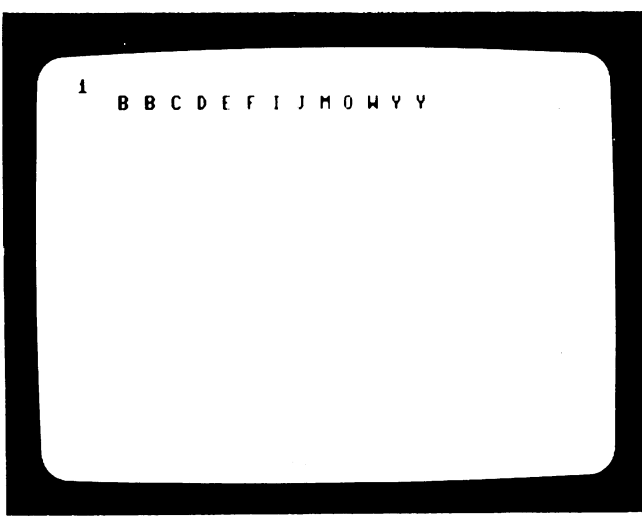
In our experience, however, any score over 200 is very good, and anything over 300 is excellent. Needless to say, the scores vary widely based on what letters you happen to get.

The program gives you a fair chance by making sure that you have at least two vowels among your 13 letters. Other than that, the letters are simply chosen at random.

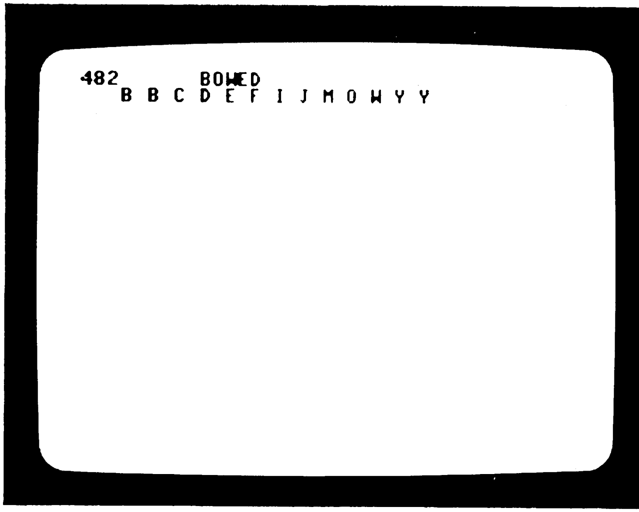
SAMPLE RUN



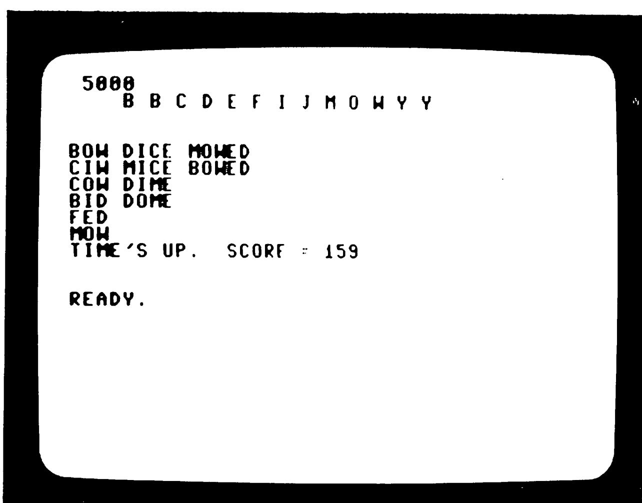
The program waits for the operator to press a key to start the game.



The program selects 13 random letters and starts the timer.



The operator enters the first word, which will go in the column of five letter words when RETURN is pressed.



The timer reaches 5000 to end the game, causing the score to be displayed. Note that a typing error was made (CIW), so the operator entered an extra three letter word. This caused the scoring of five words of three letters to be correct.

PROGRAM LISTING

```
READY.

100 REM: ARGO
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 R = RND(-TI)
140 CLR:PRINT CHR$(147):N = 13
150 DIM W$(7,6), A(N), E(N), V(5)
160 V(1) = 65:V(2) = 69:V(3) = 73
165 V(4) = 79:V(5) = 85
170 D(3) = 160:D(4) = 164:D(5) = 169
175 D(6) = 175:D(7) = 182
180 GOSUB 6000
190 GOSUB 900:PRINT CHR$(147):C = 0:M = 5000
195 D = 10:GOTO 5010
200 W$=""
210 GET A$:C = C+1
215 IF C > M THEN 810
220 NU = C:GOSUB 1000:IF A$ = "" THEN 210
230 IF ASC(A$) = 13 THEN 300
260 IF ASC(A$) = 157 THEN 1500
270 IF A$ < "A" OR A$ > "Z" THEN 210
280 W$ = W$ + A$:WO$ = W$:WL = D:GOSUB 1200
285 IF LEN(W$) > 8 THEN 300
290 GOTO 210
300 L = LEN(W$)
310 IF L < 3 OR L > 7 THEN 400
320 GOSUB 3010
330 IF F = 1 THEN M$ = "DUPLICATE":GOTO 700
340 GOSUB 4010:IF F = 1 THEN 400
350 T(L) = T(L)+1
360 IF T(L) > 6 THEN 500
370 W$(L,T(L)) = W$
380 WO$ = W$:WL = D(L):GOSUB 1200
390 WO$ = "        ":WL = D:GOSUB 1200
395 D(L) = D(L) + 40:GOTO 200
400 M$ = "ILLEGAL":GOTO 700
500 T(L) = T(L)-1:M$ = "TOO MANY"
510 GOTO 700
700 WO$ = M$:WL = D+10:GOSUB 1200
710 FOR J = 1 TO 30:C = C+1
720 GOSUB 1000:NEXT
730 WO$ = "           ":WL = D+10
735 GOSUB 1200
740 GOTO 200
810 WO$ = "TIME'S UP.  ":WL = 400:GOSUB 1200
820 SC = 0:FOR J = 3 TO 7
830 K = T(J):IF K > 5 THEN K = 5
```

```
840 SC = SC+J*J*K:NEXT
850 PRINT "SCORE =";SC
855 PRINT
860 END
900 FOR J = 1 TO N:R = INT(RND(1)*25)+65
910 A(J) = R:NEXT
920 FOR J = 1 TO 2
930 A(J) = V(INT(RND(1)*5)):NEXT
940 RETURN
1000 PRINT CHR$(19):PRINT NU:RETURN
1200 PRINT CHR$(19)
1205 XA = INT(WL/40):YA = INT(WL-(XA*40))
1210 IF YA = 0 THEN 1230
1215 FOR AY = 1 TO YA
1220 PRINT CHR$(29);:NEXT
1230 IF XA = 0 THEN 1250
1240 FOR AX = 1 TO XA
1245 PRINT CHR$(17);:NEXT
1250 PRINT WO$;
1260 RETURN
1500 IF LEN(W$) < 2 THEN 730
1510 W$ = LEFT$(W$,LEN(W$)-1)
1520 WO$ = " ":WL = D:GOSUB 1200
1530 WO$ = W$:WL = D:GOSUB 1200:GOTO 210
3010 F = 0:IF T(L) = 0 THEN RETURN
3020 FOR J = 1 TO T(L)
3030 IF W$(L,J) = W$ THEN F = 1
3040 NEXT:RETURN
4010 F = 0:FOR J = 1 TO N:E(J) = A(J)
4020 NEXT:FOR J = 1 TO L
4030 K = ASC(MID$(W$,J,1))
4040 FOR X = 1 TO N
4050 IF E(X) = K THEN E(X) = 0:GOTO 4070
4060 NEXT:F = 1
4070 NEXT J:RETURN
5010 FOR J = N TO 2 STEP -1:X = A(1):F = 1
5020 FOR L = 2 TO J:IF A(L) > X THEN X = A(L):F=L
5030 NEXT:A(F) = A(J):A(J) = X:NEXT
5050 FOR J = 1 TO N
5055 WO$ = CHR$(A(J)):WL = 42+J+J:GOSUB 1200
5100 NEXT:GOTO 730
6000 PRINT TAB(12);"A R G O"
6010 PRINT:PRINT"PRESS ANY KEY"
6020 J = INT(RND(1)*1)+1:GET A$
6030 IF A$ = "" THEN 6020
6040 RETURN


READY.
```

EASY CHANGES

1. You can easily change the program to give you more or less than 13 letters to choose from. You can use any number from three to 17, but more than 15 causes the last ones to extend to a second line. Values from nine to 15 are best. As an example, make this change to use 15 letters:

   140 CLR:PRINT CHR$(147):N=15

2. The program currently guarantees at least two vowels among the list of letters. Change the "2" at the end of line 920 to alter this. For example, to guarantee at least three vowels, it should be:

   920 FOR J=1 TO 3

3. Give the player more or less time to create words by changing the value of M in line 190. For example, to make each game last about twice as long, make this change:

   190 GOSUB 900:PRINT CHR$(147):C=0:M=10000

MAIN ROUTINES

140- 195	Initializes variables, displays title, chooses letters.
200- 290	Gets word from player. Increments timer while waiting.
300- 510	Examines word for legality. Saves it.
700- 740	Displays and erases error message.
810- 860	Computes score and ends program.
900- 940	Subroutine to select N letters.
1500-1530	Backspaces during word entry.
3010-3040	Subroutine to check for duplicate word.
4010-4070	Subroutine to check that legal letters were used.
5010-5070	Alphabetizes and displays letters.
6000-6040	Subroutine to display title and initialize RND.

MAIN VARIABLES

N	Number of letters to choose from.
W$	Array that words are saved in.
A	Array holding ASCII values of letters.
E	Array for evaluating whether legal letters were used.

V	Array with ASCII values of the vowels.
D	Array of screen locations of each word length.
C	Counter for timer.
M	Maximum value for timer.
D	Screen location for word being entered.
W$	Word being entered.
A$	Key pressed during word entry.
L	Length of word entered. Also work variable.
F	Flag set to 1 if word is illegal. Also work variable.
M$	Error message.
T	Array to count the number of words entered of each length.
J,X	Loop and work variables.
R	Random number used in selecting letters.

SUGGESTED PROJECTS

1. Display the score as each word is entered, so the player can see the score while the game is in progress.
2. Allow the player to erase the last word entered, in case of typographical error.

DECODE

PURPOSE

Decode is really more of a puzzle than a game, although you can still compete with your friends to see who can solve the puzzles the fastest. Each time you play, you are presented with a new puzzle to solve.

The object is to figure out the computer's secret code in as few guesses as possible. The program gives you information about the accuracy of each of your guesses. By carefully selecting your guesses to make use of the information you have, you can determine what the secret code must be in a surprisingly small number of guesses. Five or six is usually enough.

The first few times you try, you will probably require quite a few more guesses than that, but with practice, you'll discover that you can learn a lot more from each guess than you originally thought.

HOW TO USE IT

The program starts off by displaying a brief introduction. Here are some more details.

The program selects a secret code for you to figure out. The code is a four digit number that uses only the digits 1 through 6. For example, your Commodore 64 might pick 6153 or 2242 as a secret code.

Your object is to guess the code in the fewest possible guesses. After each of your guesses, the program tells you a "black" and a "white" number. The black number indicates the number of

digits in your guess that were correct—the digit was correct *and* in the correct position. So, if the secret code is 6153 and your guess is 4143, you will told that black is 2 (because the 1 and the 3 will have been correct). Of course, you aren't told *which* digits are correct. That is for you to figure out by making use of the information you get from other guesses.

Each of the white numbers indicates a digit in your guess that was correct, but which is in the wrong position. For example, if the secret code is 6153 and your guess is 1434, you will be told that white is 2. The 1 and 3 are correct, but in wrong positions.

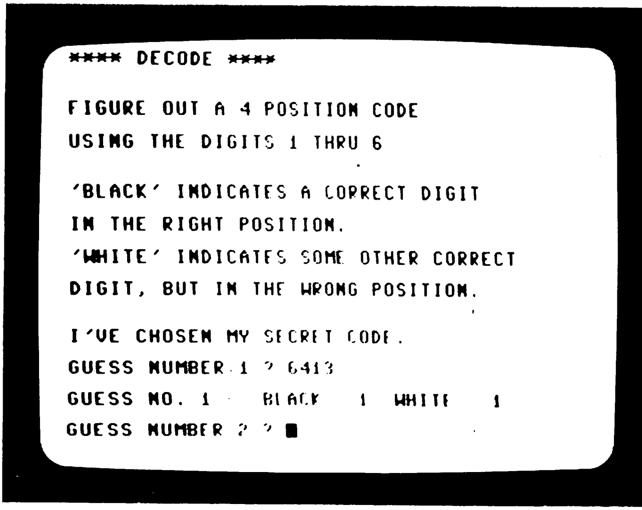
The white number is determined by ignoring any digits that accounted for a black number. Also, a single position in the secret code or guess can only account for one black or white number. These facts become significant when the secret code and/or your guess have duplicate digits. For example, if the code is 1234 and your guess is 4444, there is only one black, and no whites. If the code is 2244 and your guess is 4122, there are no blacks and three whites.

This may sound a little tricky, but you will quickly get the hang of it.

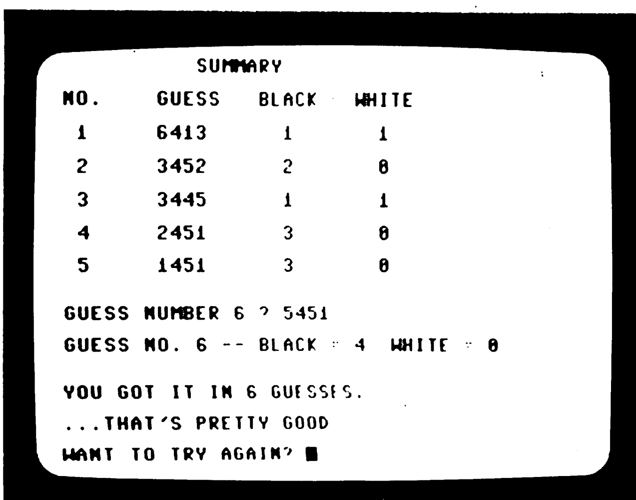
At any time during the game, you can ask for a "SUMMARY" by entering an **S** instead of a guess. This causes the program to clear the screen and display each guess (with the corresponding result) that has occurred so far.

Also, if you get tired of trying and want to give up, you can enter a **Q** (for "quit") to end your misery and find out the answer. Otherwise, you continue guessing until you get the code right (four black, zero white), or until you have used up the maximum of twelve guesses.

SAMPLE RUN



The program displays an introduction, chooses its secret code, and asks for the operator's first guess. After the operator makes a guess, the program responds with a "black" and a "white" number, and asks for the second guess.



Later in the same game, the operator asks for a summary, then makes the guess that turns out to be correct. The program acknowledges that the guess is correct and asks about trying another game.

PROGRAM LISTING

```
READY.

100 REM: DECODE
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 D=6:P=4:L=12
130 DIM G$(L),G(P),C(P),B(L),W(L)
140 R=RND(-TI)
150 GOSUB 1200
170 GOSUB 300:GOSUB 370
180 PRINT"GUESS NUMBER";G;
190 INPUT A$
200 IF LEFT$(A$,1)="S" THEN 500
210 IF LEFT$(A$,1)="Q" THEN 600
220 GOSUB 700
230 GOSUB 800
240 GOSUB 1000
250 IF B(G)=P THEN 2000
260 G$(G)=A$
270 G=G+1:IF G>L THEN 2200
280 GOTO 180
300 G=1:C$=""
310 RETURN
370 FOR J=1 TO P
380 R=INT(D*RND(1))+1
390 C$=C$+MID$(STR$(R),2,1)
400 NEXT J
410 PRINT"I'VE CHOSEN MY SECRET CODE."
420 PRINT
430 RETURN
500 IF G=1 THEN PRINT"NO GUESSES YET":GOTO 180
510 PRINT CHR$(147),"SUMMARY":PRINT
520 PRINT"NO.     GUESS    BLACK    WHITE"
530 PRINT:FOR J=1 TO G-1
540 PRINT J;TAB(7);G$(J);TAB(16);B(J);TAB(24);W(J)
550 IF G<10 THEN PRINT
560 NEXT:PRINT
570 GOTO 180
600 PRINT
610 PRINT"CAN'T TAKE IT, HUH?"
620 PRINT:PRINT"WELL, MY CODE WAS ";
630 FOR J=1 TO 4
640 PRINT" .";
650 FOR K=1 TO 300:NEXT
660 NEXT
670 PRINT C$:PRINT
```

```
680 GOTO 2090
700 IF LEN(A$)<>P THEN 780
710 FOR J=1 TO P
720 R=VAL(MID$(A$,J,1))
730 IF R<1 OR R>D THEN 780
740 NEXT
750 RETURN
780 PRINT"ILLEGAL.  TRY AGAIN."
790 GOTO 180
800 B=0:W=0
810 FOR J=1 TO P
820 G(J)=VAL(MID$(A$,J,1))
830 C(J)=VAL(MID$(C$,J,1))
840 IF G(J)=C(J) THEN B=B+1:G(J)=0:C(J)=0
850 NEXT
860 FOR J=1 TO P:IF C(J)=0 THEN 920
870 H=0:FOR K=1 TO P
880 IF C(J)=0 THEN 910
890 IF C(J)<>G(K) THEN 910
900 H=1:G(K)=0:C(J)=0
910 NEXT K:W=W+H
920 NEXT J
930 RETURN
1000 B(G)=B:W(G)=W:PRINT
1010 PRINT"GUESS NO.";G;"-- BLACK =";B;
1015 PRINT" WHITE =";W
1020 PRINT:RETURN
1200 PRINT CHR$(147);
1210 PRINT"**** DECODE ****"
1220 PRINT:PRINT
1230 PRINT"FIGURE OUT A";P;"POSITION CODE"
1240 PRINT
1250 PRINT"USING THE DIGITS 1 THRU";D
1260 PRINT:PRINT
1270 PRINT"'BLACK' INDICATES A CORRECT DIGIT"
1280 PRINT:PRINT"IN THE RIGHT POSITION."
1290 PRINT
1300 PRINT"'WHITE' INDICATES SOME OTHER CORRECT"
1310 PRINT
1320 PRINT"DIGIT, BUT IN THE WRONG POSITION."
1330 PRINT:PRINT
1340 RETURN
2000 PRINT
2010 PRINT"YOU GOT IT IN";G;"GUESSES."
2020 IF G<5 THEN B$="OUTSTANDING!"
2030 IF G=5 OR G=6 THEN B$="PRETTY GOOD"
2040 IF G=7 THEN B$="NOT BAD"
2050 IF G=8 THEN B$="NOT TOO GREAT"
2060 IF G>8 THEN B$="PRETTY BAD"
2070 PRINT:PRINT"...THAT'S ";B$
```

```
2080 PRINT
2090 INPUT"WANT TO TRY AGAIN";A$
2100 IF LEFT$(A$,1)="Y" THEN 150
2110 IF LEFT$(A$,1)<>"N" THEN 2090
2120 PRINT:PRINT"COWARD.":PRINT
2130 END
2200 PRINT
2210 PRINT"THAT'S YOUR LIMIT OF";L;"GUESSES."
2220 PRINT
2230 PRINT"MY CODE WAS ";C$
2240 GOTO 2080

READY.
```

EASY CHANGES

1. Modify line 120 to change the complexity of the code and/or the number of guesses you are allowed. For example, the following line would allow fifteen guesses at a five position code using the digits 1 through 8:

   120 D=8:P=5:L=15

   The introduction will automatically reflect the new values for D and P. Be sure that neither D nor P is set greater than 9.
2. To change the program so it will always display the "Summary" information after each guess automatically, replace line 280 with this:

   280 GOTO 500

MAIN ROUTINES

120- 170	Initializes variables. Displays introduction. Chooses secret code.
180- 240	Gets a guess from operator. Analyzes reply. Displays result.
250	Determines if operator guessed correctly.
260- 280	Saves guess. Adds one to guess counter. Determines if limit on number of guesses was exceeded.
300- 310	Subroutine to initialize variables, choose secret code and inform operator.
500- 570	Subroutine to display summary of guesses so far.
600- 680	Subroutine to slowly display secret code when operator quits.

700- 790	Subroutine to determine if operator's guess was legal.
800- 930	Subroutine to determine number of black and white responses for the guess.
1000-1020	Subroutine to display number of black and white responses for guess.
1200-1340	Subroutine to display title and introduction.
2000-2130	Subroutine to analyze operator's performance after correct answer is guessed and ask about playing again.
2200-2240	Subroutine to display secret code after operator exceeds limit of number of guesses.

MAIN VARIABLES

D	Number of possible digits in each position of the code (i.e., a digit from 1 to D).
P	Number of positions in the code.
L	Limit of number of guesses that can be made.
G$	Array in which guesses are saved.
G,C	Work arrays in which each guess is analyzed.
B,W	Arrays in which the number of black and white responses is saved for each guess.
R,H	Work variables.
G	Counter of the number of guesses made.
A$	Reply by the operator.
C$	Secret code chosen by the program.
J,K	Loop variables.
B,W	Number of black and white responses for this guess.
B$	String with message about operator's performance.

SUGGESTED PROJECTS

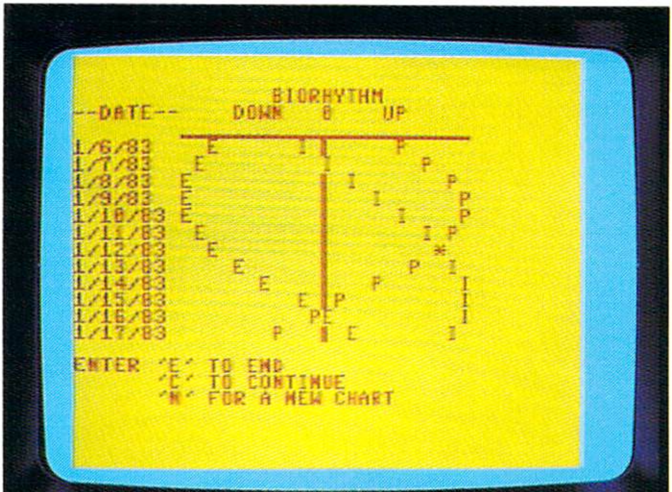
1. Change the analysis at the end of the game to take into account the difficulty of the code as well as the number of guesses it took to figure the code out. A four position code using the digits 1 through 6 has 1296 possibilities, but a five position code using 1 through 8 has 32768 possibilities. Change lines 2020 through 2060 to determine the message to be displayed based on the number of possibilities in the code as well as G.

2. At the beginning of the game, give the operator the option of deciding the complexity of the code. Ask for the number of positions and the number of digits. Make sure only "reasonable" numbers are used—do not try to create a code with zero positions, for example. Another approach is to ask the operator if he/she wants to play the easy, intermediate, or advanced version. Then set the values of D and P accordingly. Suggestions are:

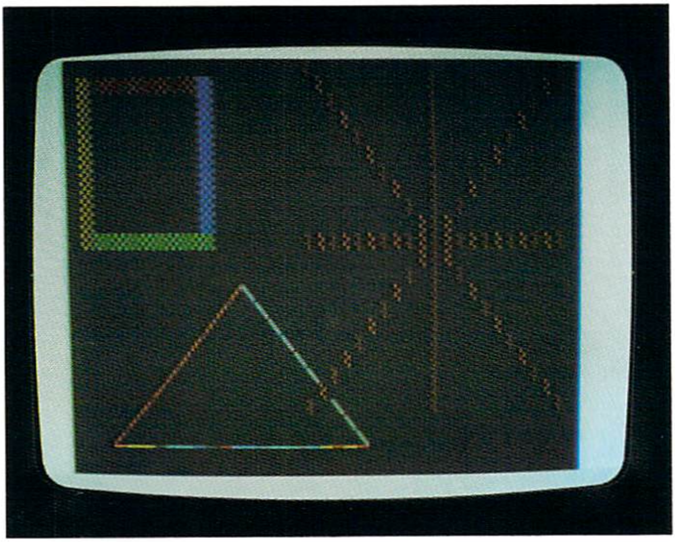
Easy:	D = 3 and P = 3
Intermediate:	D = 6 and P = 4
Advanced:	D = 8 and P = 5

3. In addition to using the number of guesses to determine how well the operator did, keep track of the amount of time. This will require use of the GET function instead of the INPUT function in line 190, and a bit of logic to "build" the A$ reply one character at a time. By counting the number of null strings encountered while waiting for keys to be pressed, you can "time" the operator.

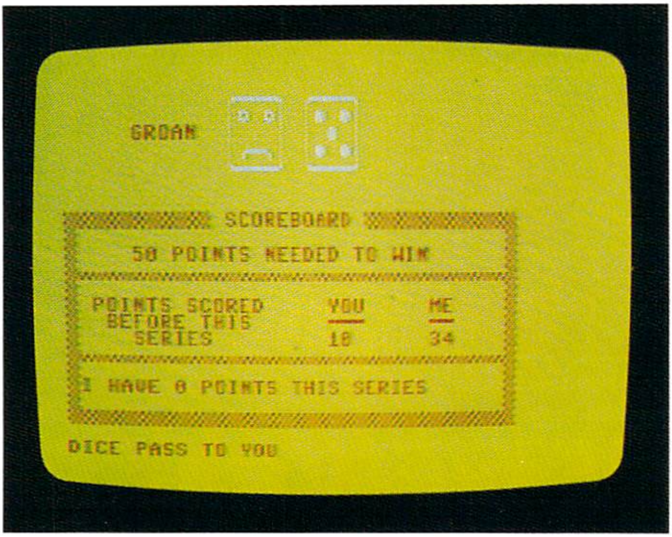
Color Section



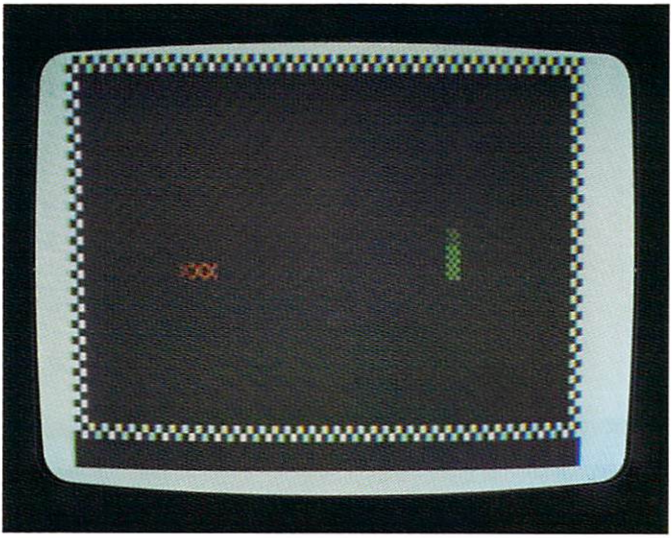
BIORHYTHM



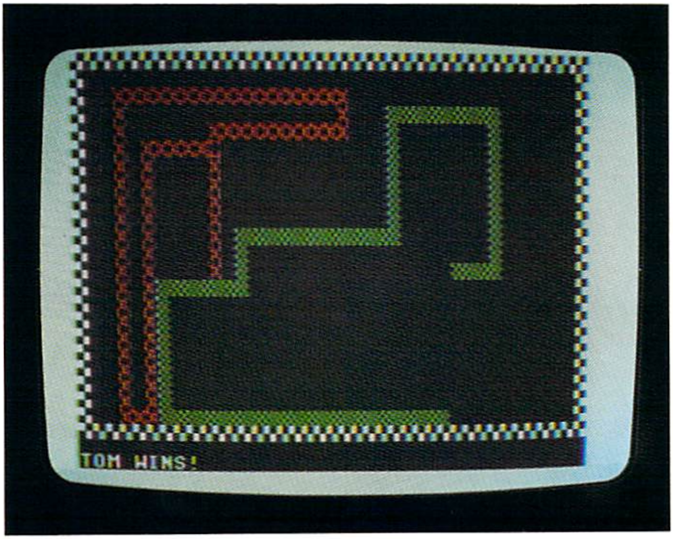
NUMBERS



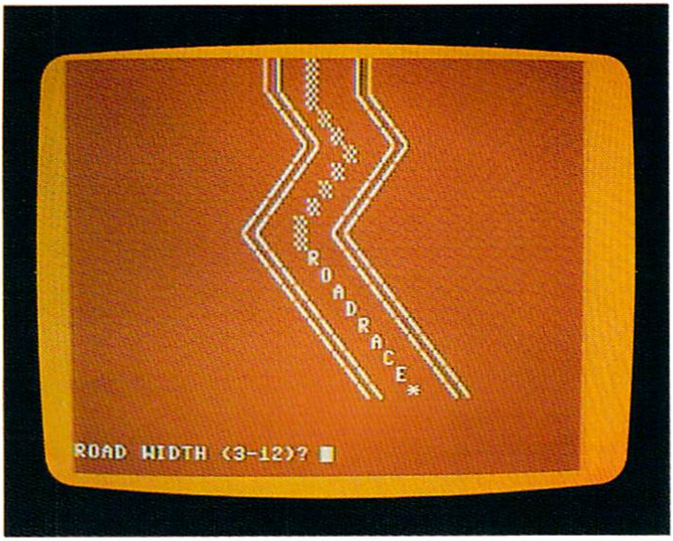
GROAN



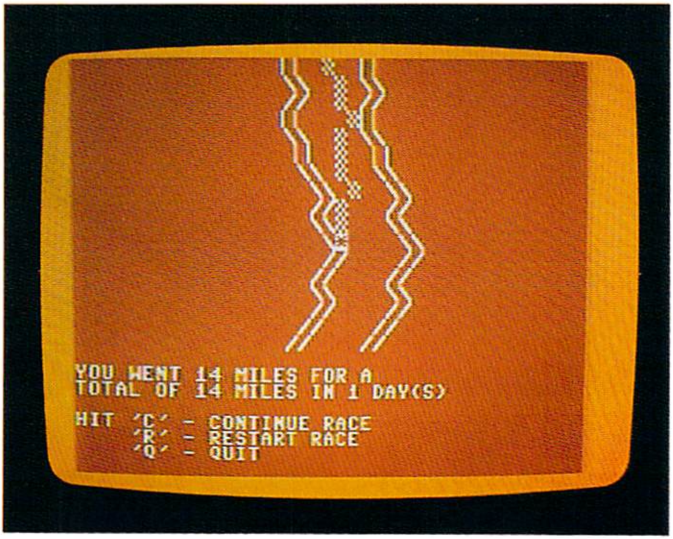
OBSTACLE



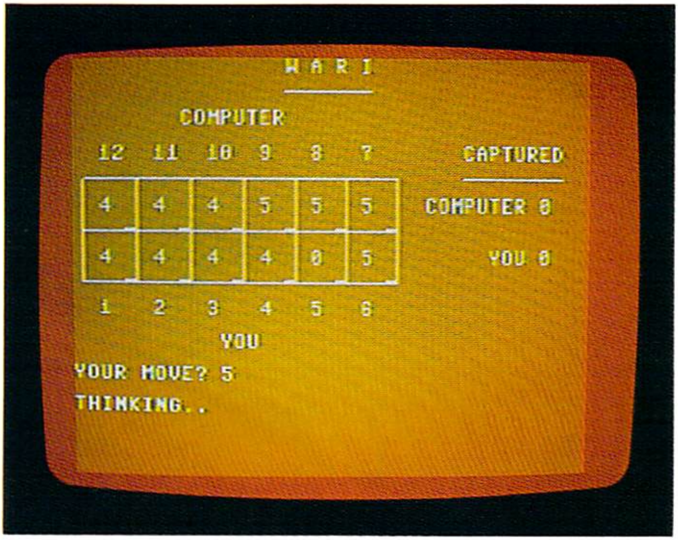
OBSTACLE



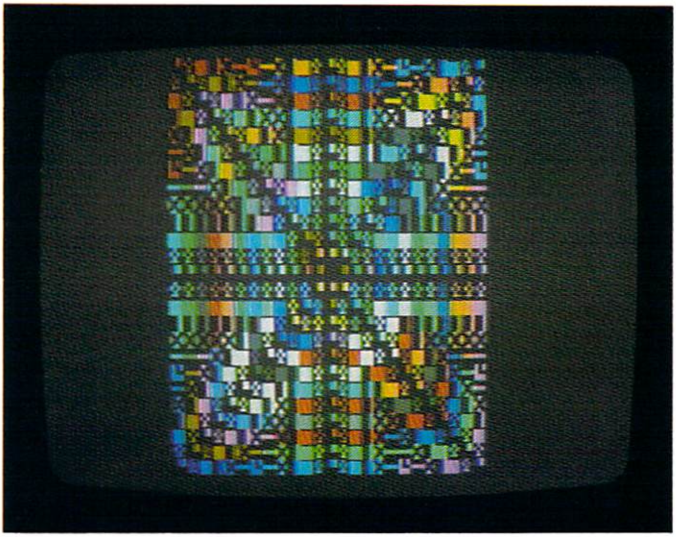
ROADRACE



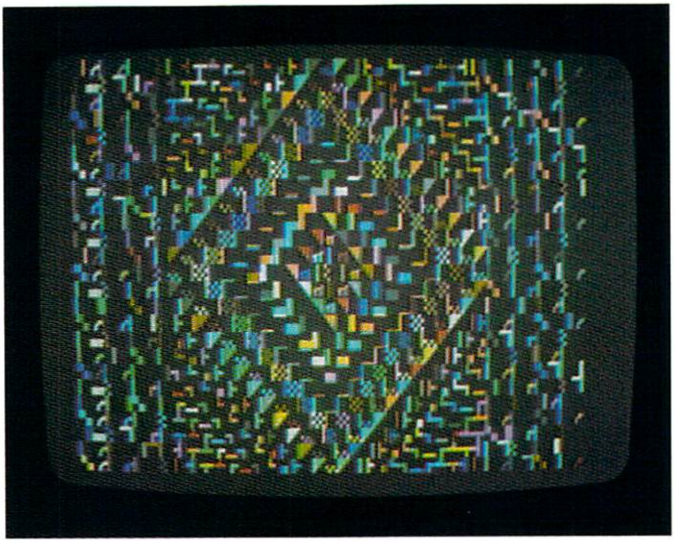
ROADRACE



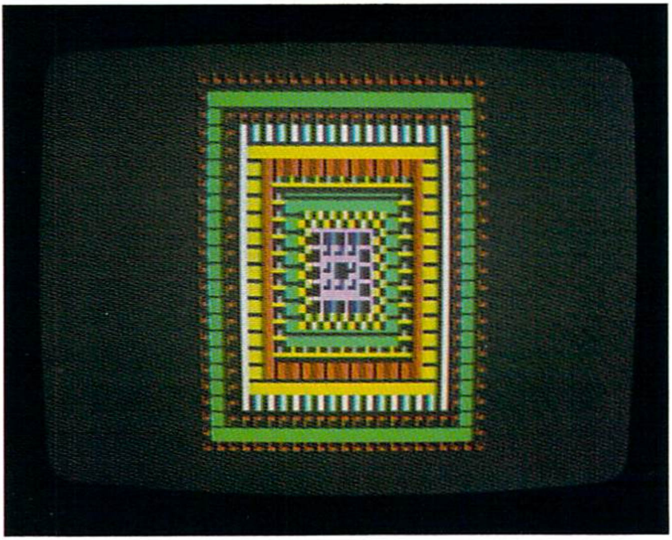
WARI



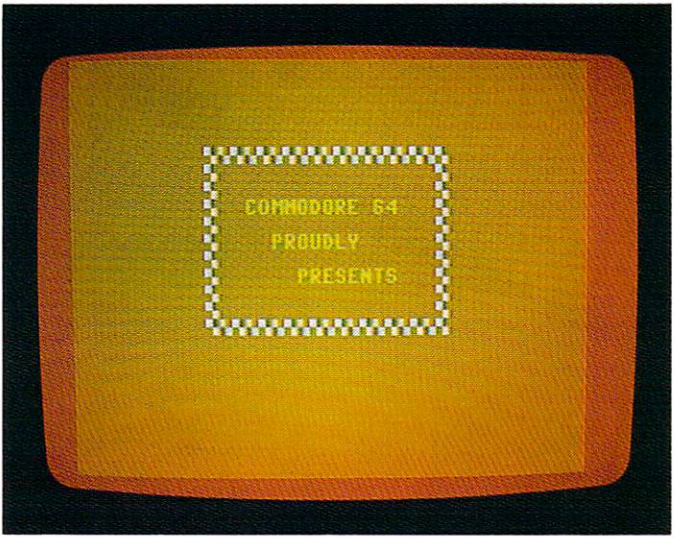
KALEIDO



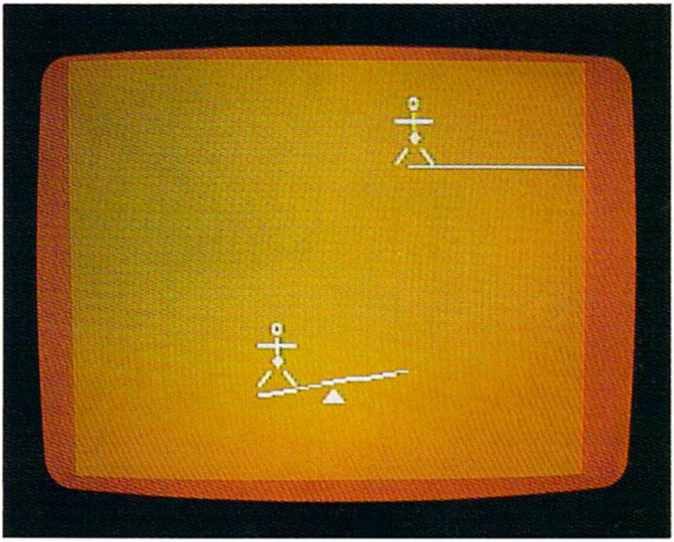
SPARKLE



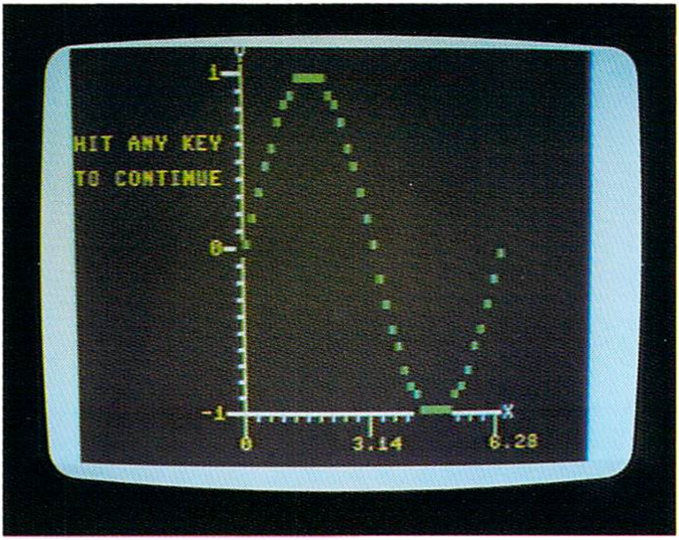
SQUARES



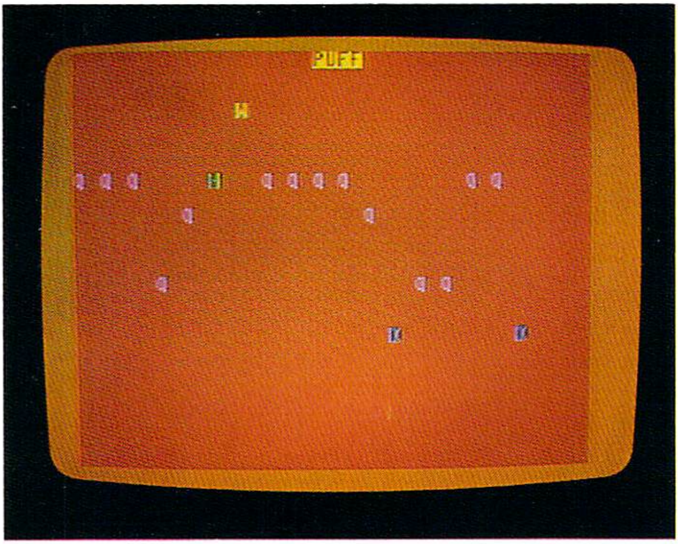
WALLOONS



WALLOONS



GRAPH



TUNE

GROAN

PURPOSE

Do you like the thrills of fast-paced dice games? If so, GROAN is right up your alley. It is a two-person game with the computer playing directly against you. There is a considerable amount of luck involved. However, the skill of deciding when to pass the dice to your opponent also figures prominently.

The Commodore 64 will roll the dice for both players, but don't worry—it will not cheat. (We wouldn't think of stooping to such depths.)

Why is the game called GROAN? You will know soon after playing it.

HOW TO USE IT

The game uses two dice. They are just like regular six-sided dice except for one thing. The die face where the "1" would normally be has a picture of a frowning face instead. The other five faces of each die have the usual numbers two through six on them.

The object is to be the first player to achieve a score agreed upon before the start of the game. Players alternate taking turns. A turn consists of a series of dice rolls (at least one roll, possibly several) subject to the following rules.

As long as no frown appears on either die, the roller builds a running score for this current series of rolls. After each roll with no frown, he has the choice of rolling again or passing the dice to his opponent. If he passes the dice, his score achieved on the current series is added to any previous total he may have had.

But if he rolls and a frown appears, he will be groaning. A frown on only one die cancels any score achieved for the current series of rolls. Any previous score is retained in this case. However, if he rolls a double frown, his entire previous total is wiped out as well as his current total. Thus, he reverts back to a total score of zero—true despair.

The program begins by asking what the winning score should be. Values between 50 and 100 tend to produce the best games, but any positive value less than 1000 is acceptable. Next, you are asked to hit any key to begin the simulated coin toss which randomly decides who will get the first roll.

Each dice roll is portrayed with a short graphics display. The dice are shown rolling and then the outcome is displayed pictorially. Before each roll, the Commodore 64 indicates whose roll is coming up.

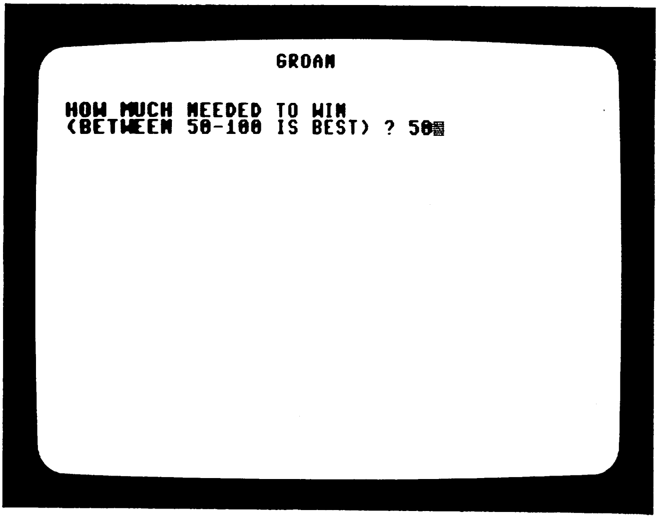
Each roll is followed by a display of the scoreboard. This scoreboard gives all relevant information: score needed to win, both players' scores before the current series of rolls, and the total score for the current series.

If a frown should appear on a die, the scoreboard will indicate the current running total as zero. In addition, the previous total will become zero in the case of the dreaded double frown. In either case, the dice will be passed automatically to the other player.

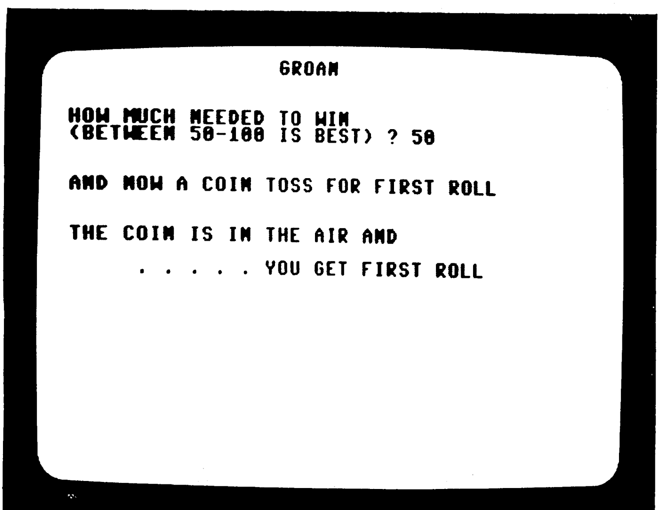
If a scoring roll results, the roller must decide whether to roll again or to pass the dice. The program has a built-in strategy to decide this for the computer. For you, the question will be asked after the scoreboard is displayed. The two legal replies are **P** and **R**. The **R** means that you wish to roll again. The **P** means that you choose to pass the dice to the computer. If you should score enough to win, you must still pass the dice to add the current series to your previous total.

The first player to pass the dice with a score greater than or equal to the winning score is the victor. This will surely cause his opponent to GROAN. The computer will acknowledge the winner before signing off.

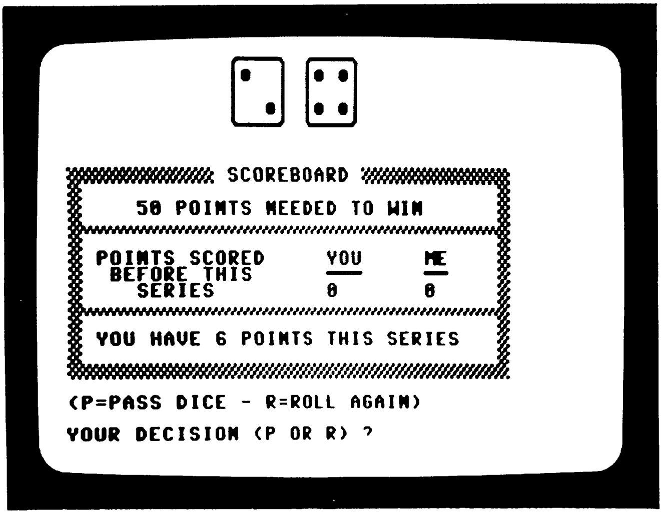
SAMPLE RUN



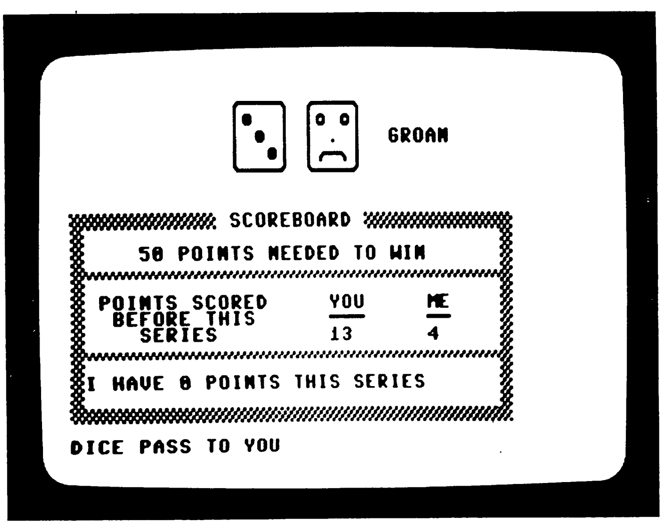
The operator has decided to challenge the computer to a fifty point game of Groan.



The operator wins the coin toss and gets the first dice roll.



The operator's roll results in a 2 and a 4 for a total of 6 points. The operator must now decide whether to pass the dice or risk rolling again.



Later in the same game, the computer rolls a three and a "groan." This scores no points for the series and the dice pass to the operator.

PROGRAM LISTING

```
READY.

100 REM: GROAN
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 Q=RND(-TI):
160 S=1024:OS=54272
170 DL=200
180 B=166:D=168:U=196
190 POKE 53280,6:POKE 53281,6
195 PRINT CHR$(5)
200 PRINT CHR$(147);TAB(16);"GROAN":PRINT
210 PRINT:PRINT"HOW MUCH NEEDED TO WIN"
215 INPUT"(BETWEEN 50-100 IS BEST) ";W
220 W=INT(W):IF W<=0 THEN 210
230 PRINT:PRINT
232 PRINT"AND NOW A COIN TOSS FOR FIRST ROLL"
234 GOSUB 830
240 PRINT:PRINT:PRINT"THE COIN IS IN THE AIR AND"
250 Q$="YOU":Q=RND(1):IF Q>.5 THEN Q$="I"
260 PRINT:PRINT SPC(5);
265 FOR J=1 TO 5:PRINT". ";:GOSUB 830:NEXT
270 PRINT Q$;" GET FIRST ROLL":GOSUB 840
275 POKE 53280,5:POKE 53281,5:PRINT CHR$(5)
280 T=0:IF Q>.5 THEN 400
300 P$=" YOU"
305 PRINT CHR$(147);TAB(11);"YOU'RE ROLLING"
308 GOSUB 830:GOSUB 500
310 T=T+R1+R2:IF F>0 THEN T=0
320 IF F=2 THEN H=0
330 GOSUB 850
333 IF F>0 THEN PRINT"DICE PASS TO ME"
336 IF F>0 THEN GOSUB 840:GOTO 400
340 PRINT"(P=PASS DICE - R=ROLL AGAIN)"
345 PRINT:PRINT"YOUR DECISION (P OR R) ?"
350 GET Q$:IF Q$="" THEN 350
360 IF Q$="R" THEN 300
370 IF Q$<>"P" THEN 350
380 PRINT:H=H+T:IF H>=W THEN 970
390 T=0:F=1:PRINT CHR$(147):GOTO 330
400 T=0:P$="I"
410 PRINT CHR$(147);CHR$(17);TAB(12);"I'M ROLLING"
415 GOSUB 830:GOSUB 500
420 T=T+R1+R2:IF F>0 THEN T=0
430 IF F=2 THEN P=0
440 GOSUB 850:IF F>0 THEN PRINT"DICE PASS TO YOU"
445 IF F>0 THEN GOSUB 840:T=0:GOTO 300
450 GOSUB 1000:IF X=1 THEN PRINT"I'LL ROLL AGAIN"
```

```
455 IF X=1 THEN GOSUB 840:GOTO 410
460 PRINT"I'LL STOP WITH THIS"
465 GOSUB 830:P=P+T:IF P>=W THEN 970
470 PRINT:PRINT"DICE PASS TO YOU"
475 T=0:GOSUB 840:GOTO 300
500 C2=S+420:DL=500:GOSUB 600:R1=INT(RND(1)*6+1)
505 R2=INT(RND(1)*6+1)
510 DL=60:C2=C2-118:GOSUB 600:C2=C2-77:GOSUB 600
515 C2=C2+83:GOSUB 600
520 C2=C2+122:GOSUB 600:C2=C2+118:GOSUB 600
525 C2=C2+77:GOSUB 600
530 C2=C2-83:GOSUB 600
540 C2=C2-122:C1=2*(S+417)-C2
550 C=C1:R=R1:GOSUB 650:GOSUB 700
560 C=C2:R=R2:GOSUB 650:GOSUB 700:F=0
565 IF R1=1 THEN F=1:NT=5:GOSUB 800
570 IF R2=1 THEN F=F+1:NT=25:GOSUB 800
580 IF F=2 THEN GOSUB 820:GOSUB 830
590 RETURN
600 C1=2*(S+417)-C2:C=C1:GOSUB 650
610 C=C2:GOSUB 650:FOR J=1 TO DL:NEXT
620 PRINT CHR$(147):RETURN
650 POKE C-82,85:POKE C-82+OS,1:POKE C-7 8,73
655 POKE C-78+OS,1:POKE C+78,74:POKE C+7 8+OS,1
660 POKE C+82,75:POKE C+82+OS,1:FOR J=1  TO 3
665 POKE C-82+J,64:POKE C-82+J+OS,
    1:POKE C-78+40*J,93
670 POKE C-78+40*J+OS,1:POKE C-82+40*J,  93
675 POKE C-82+40*J+OS,1:POKE C+78+J,64
680 POKE C+78+J+OS,1:NEXT:RETURN
700 ON R GOSUB 705,725,735,740,750,755:  RETURN
705 POKE C-41,87:POKE C-41+OS,1:POKE C-3 9,87
710 POKE C-39+OS,1:POKE C,46:POKE C+OS,1
715 POKE C+39,85:POKE C+39+OS,1:POKE C+4 0,64
720 POKE C+40+OS,1:POKE
    C+41,73:POKE C+4 1+OS,1:RETURN
725 POKE C-41,81:POKE C-41+OS,1:POKE C+  41,81
730 POKE C+41+OS,1:RETURN
735 POKE C,81:POKE C+OS,1:GOSUB 725:     RETURN
740 GOSUB 725:POKE C-39,81:POKE C-39+OS, 1
745 POKE C+39,81:POKE C+39+OS,1:RETURN
750 GOSUB 740:POKE C,81:POKE C+OS,1:     RETURN
755 GOSUB 740:POKE C-1,81:POKE C-1+OS,1
760 POKE C+1,81:POKE C+1+OS,1:RETURN
800 PRINT CHR$(19)
805 FOR J=1 TO 9:PRINT CHR$(17);:NEXT
810 PRINT TAB(NT);"GROAN":GOSUB 830:RETURN
820 FOR J=1 TO 5:PRINT CHR$(145);:NEXT
825 PRINT TAB(14);CHR$(18);"DESPAIR":RETURN
830 FOR K=1 TO 1500:NEXT:RETURN
840 FOR K=1 TO 5000:NEXT:RETURN
850 PRINT CHR$(19)
```

```
853 FOR J=1 TO 13:PRINT CHR$(17);:NEXT
856 PRINT:FOR J=1 TO 11
860 PRINT CHR$(B);:NEXT
863 PRINT" SCOREBOARD ";
866 FOR J=1 TO 12:PRINT CHR$(B);:NEXT
868 PRINT
870 GOSUB 960:PRINT CHR$(B);SPC(S);W;
875 PRINT "POINTS NEEDED TO WIN";TAB(34);CHR$(B)
880 PRINT CHR$(B);
883 FOR J=1 TO 33:PRINT CHR$(D);:NEXT
886 PRINT CHR$(B):GOSUB 960
890 PRINT CHR$(B);" POINTS SCORED      YOU      ME";
895 PRINT TAB(34);CHR$(B)
900 PRINT CHR$(B);"  BEFORE THIS";
905 PRINT TAB(20);CHR$(U);CHR$(U);CHR$(U);
910 PRINT TAB(28);CHR$(U);CHR$(U);TAB(34);CHR$(B)
920 PRINT CHR$(B);"     SERIES";TAB(19);H;
925 PRINT TAB(27);P;TAB(34);CHR$(B)
930 PRINT CHR$(B);
933 FOR J=1 TO 33:PRINT CHR$(D);:NEXT
936 PRINT CHR$(B):GOSUB 960
940 PRINT CHR$(B);P$;
943 PRINT " HAVE";T;"POINTS THIS SERIES";
946 PRINT TAB(34);CHR$(B):GOSUB 960
950 FOR J=1 TO 35:PRINT CHR$(B);:NEXT
955 PRINT:PRINT:RETURN
960 PRINT CHR$(B);TAB(34);CHR$(B):RETURN
970 T=0:PRINT CHR$(147):GOSUB 850
975 IF P>=W THEN PRINT:PRINT"SKILL WINS AGAIN"
980 IF H>=W THEN PRINT"YOU WIN -";
965 IF H>=W THEN PRINT" IT WAS SHEER LUCK"
990 POKE 53280,14:POKE 53281,6
995 END
1000 V=P+T:IF V>=W THEN 1100
1010 IF (W-H)<10 THEN 1110
1020 IF P>=H THEN L=T/25:GOTO 1050
1030 IF V<H THEN L=T/35:GOTO 1050
1040 L=T/30
1050 IF RND(1)>L THEN 1110
1100 X=0:RETURN
1110 X=1:RETURN

READY.
```

EASY CHANGES

1. If you wish to set the program for a fixed value of the winning score, it can be done by deleting line 210 and changing line 215. Simply set W to the winning score desired. For example:

```
215 W=100
```

would make the winning score 100.

2 . The rolling dice graphics display before each roll can be eliminated by adding line 507 as follows:

```
507 GOTO 550
```

This has the effect of speeding up the game by showing each dice roll immediately.

3. After you play the game a few times, you may wish to change the delay constants in lines 830 and 840. They control the "pacing" of the game; i.e., the time delays between various messages, etc. To speed up the game try

```
830 FOR K=1 TO  500:NEXT:RETURN
840 FOR K=1 TO 1000:NEXT:RETURN
```

Of course, if desired, the constants can be set to larger values to slow down the pacing.

MAIN ROUTINES

150- 180	Initializes constants.
200- 280	Initial display. Gets winning score.
300- 390	Human rolls.
400- 475	Commodore 64 rolls.
500- 620	Determines dice roll, drives its display.
650- 760	Draws die face and outline.
800- 825	Displays groan messages.
830- 840	Delay loops.
850- 960	Displays scoreboard.
970- 990	Ending messages.
1000-1110	Computer's strategy. Sets X=0 to stop rolling or X=1 to continue rolling.

MAIN VARIABLES

W	Amount needed to win.
H	Previous score of human.
P	Previous score of computer.
T	Score of current series of rolls.
X	Computer strategy flag (0=stop rolling; 1=roll again).
L	Cutoff threshold used in computer's built-in strategy.

V	Score computer would have if it passed the dice.
Q,Q$	Work variable, work string variable.
J,K	Loop indices.
P$	String of name of current roller.
R1,R2	Outcome of roll for die 1, die 2.
R	Outcome of a die roll.
F	Result of roll (0 = no frown; 1 = one frown; 2 = double frown).
B,D,U	CHR$ arguments for border, divider, underline.
S	Starting argument for CRT displays.
C1,C2	POKE address of die 1, die 2.
C	POKE address of a die.
NT	Argument for TAB function.
DL	Delay Length

SUGGESTED PROJECTS

1. The computer's built-in strategy is contained from line 1000 on. Remember, after a no frown roll, the Commodore 64 must decide whether or not to continue rolling. See if you can improve on the current strategy. You may use, but not modify, the variables P, T, H, W. The variable X must be set before returning. Set X = 0 to mean the computer passes the dice or X = 1 to mean the computer will roll again.
2. Ask the operator for his/her name. Then personalize the messages and scoreboard more.
3. Dig into the workings of the graphics routines connected with the dice rolling. Then modify them to produce new, perhaps more realistic, effects.

JOT

PURPOSE

JOT is a two player word game involving considerable mental deduction. The Commodore 64 will play against you. But be careful! You will find your computer quite a formidable opponent.

The rules of JOT are fairly simple. The game is played entirely with three-letter words. All letters of each word must be distinct—no repeats. (See the section on Easy Changes for further criteria used in defining legal words.)

To begin the game, each player chooses a secret word. The remainder of the game involves trying to be the first player to deduce the other's secret word.

The players take turns making guesses at their opponent's word. After each guess, the asker is told how many letters (or hits) his guess had in common with his opponent's secret word. The position of the letters in the word does not matter. For example, if the secret word was "own," a guess of "who" would have 2 hits. The winner is the first person to correctly guess his opponent's secret word.

HOW TO USE IT

The program begins with some introductory messages while asking you to think of your secret word. It then asks whether or not you wish to make the first guess. This is followed by you and the Commodore 64 alternating guesses at each other's secret word.

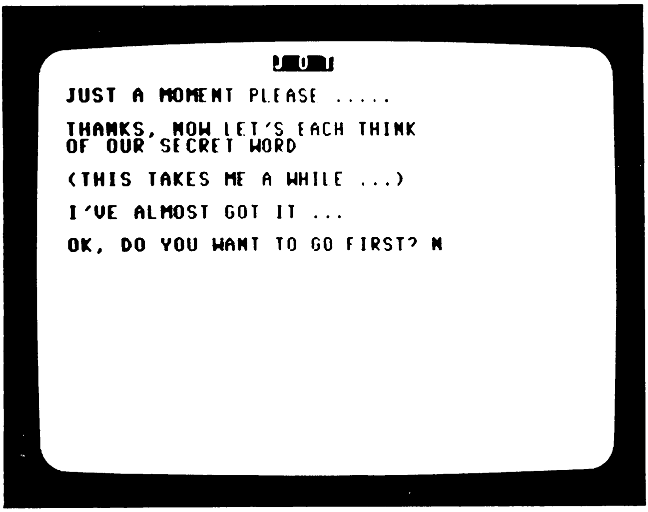
After the computer guesses, it will immediately ask you how it did. Possible replies are **0, 1, 2, 3,** or **R**. The response of **R** (for right) means the Commodore 64 has just guessed your word correctly—a truly humbling experience. The numerical replies indicate that the word guessed by the computer had that number of hits in your secret word. A response of **3** means that all the letters were correct, but they need to be rearranged to form the actual secret word (e.g. a guess of "EAT" with the secret word being "TEA").

After learning how it did, the computer will take some time to process its new information. If this time is not trivial, the Commodore 64 will display the message: "I'M THINKING" so you do not suspect it of idle daydreaming. If it finds an inconsistency in its information, it will ask you for your secret word and then analyze what went wrong.

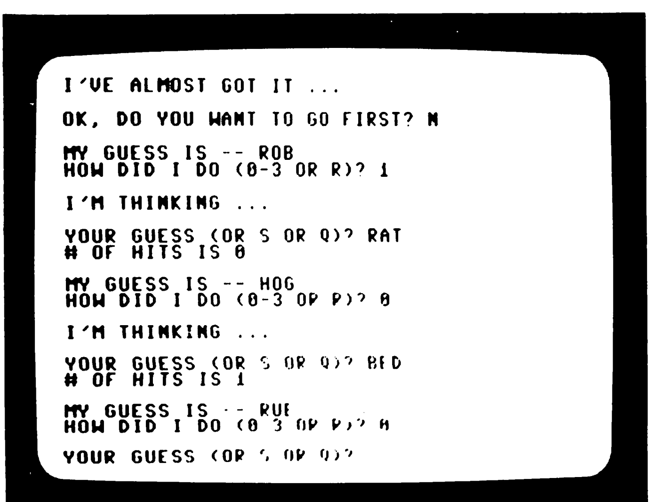
When it is your turn to guess, there are two special replies you can make. These are the single letters **S** or **Q**. The **S**, for summary, will display a table of all previous guesses and corresponding hits. This is useful as a concise look at all available information. It will then prompt you again for your next guess. The **Q**, for quit, will simply terminate the game.

When not making one of these special replies, you will input a guess at the computer's secret word. This will be, of course, a three letter word. If the word used is not legal, the computer will so inform you. After a legal guess, you will be told how many hits your guess had. If you correctly guess the computer's word, you will be duly congratulated. The computer will then ask you for your secret word and verify that all is on the "up and up."

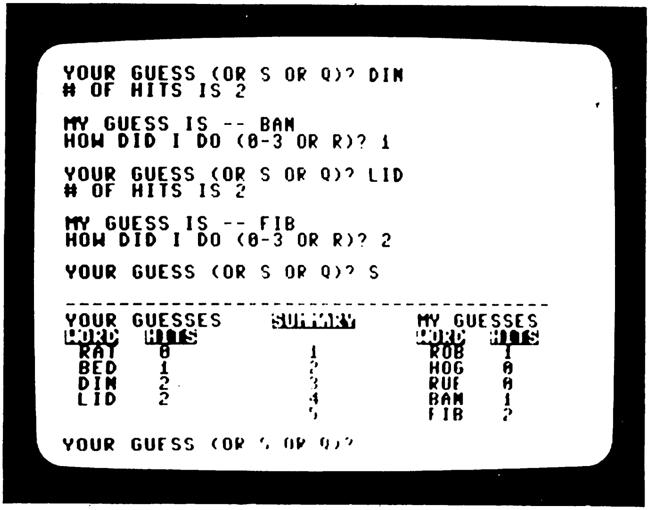
SAMPLE RUN



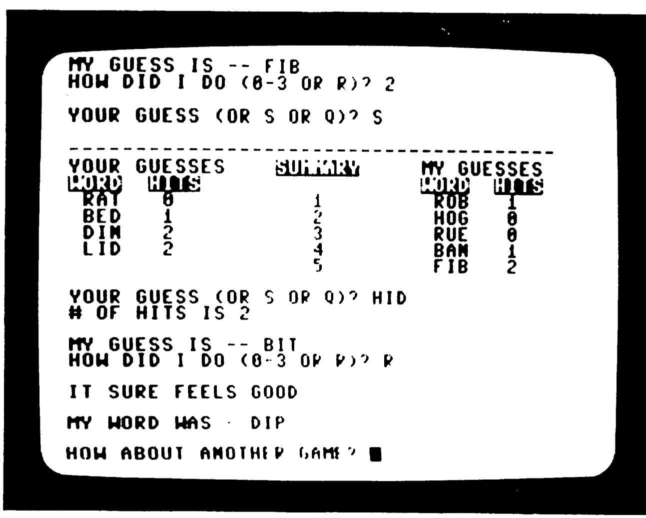
The player and the computer each select their secret words. The computer is given the first guess.



The computer and player exchange the first few guesses and their results with each other.



Later in the same game, the player requests a summary before making his guess.



The computer, however, guesses correctly to win the game. After revealing its secret word, the computer offers another game but the player has had enough.

PROGRAM LISTING

```
READY.

100 REM: JOT
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 M=25:N=406:V=250
160 DIM A$(V),B$(N-V)
170 DIM G1$(M),G2$(M),H1(M),H2(M)
200 G1=0:G2=0
210 L=N:Q=RND(-TI)
250 PRINT CHR$(147);SPC(16);CHR$(18);"J O T":PRINT
260 PRINT"JUST A MOMENT PLEASE ....."
265 GOSUB 3000:PRINT:Q=RND(1)*N+1
270 PRINT"THANKS, NOW LET'S EACH THINK"
275 PRINT"OF OUR SECRET WORD"
280 PRINT:PRINT"(THIS TAKES ME A WHILE ...)"
290 GOSUB 2200:GOSUB 2000:M$=Q$:PRINT:PRINT"OK, ";
300 INPUT"DO YOU WANT TO GO FIRST";Q$
310 Q$=LEFT$(Q$,1):IF Q$="N" THEN 600
320 IF Q$="Y" THEN 500
330 PRINT:PRINT"YES OR NO PLEASE":PRINT:GOTO 300
500 PRINT:INPUT"YOUR GUESS (OR S OR Q)";P$
505 IF P$="S" THEN GOSUB 1000:GOTO 500
510 IF P$="Q" THEN 1100
520 IF P$=M$ THEN G1=G1+1:G1$(G1)=P$
525 IF P$=M$ THEN H1(G1)=9:GOTO 3400
530 GOSUB 1800
533 IF F=0 THEN PRINT"THAT'S NOT A LEGAL WORD";
536 IF F=0 THEN PRINT" -- TRY AGAIN":GOTO 500
540 Q$=M$:GOSUB 2600:Q$=P$:GOSUB 1500
550 PRINT"# OF HITS IS";Q
560 G1=G1+1:G1$(G1)=Q$:H1(G1)=Q
570 IF G1=M THEN 3600
600 Q=L:GOSUB 2000:G2=G2+1:G2$(G2)=Q$
610 PRINT:PRINT"MY GUESS IS -- ";Q$
620 INPUT"HOW DID I DO (0-3 OR R)";P$
630 P$=LEFT$(P$,1)
640 IF P$="R" THEN H2(G2)=9:GOTO 3200
650 P=VAL(P$)
653 IF P>3 OR (P=0 AND P$<>"0") THEN GOSUB 659
656 IF P>3 OR (P=0 AND P$<>"0") THEN GOTO 610
658 GOTO 660
659 PRINT "BAD ANSWER":RETURN
660 IF L>100 THEN PRINT:PRINT"I'M THINKING ..."
670 H2(G2)=P:GOSUB 800
```

```
680 GOTO 500
800 Q$=G2$(G2):H=H2(G2):J=0:GOSUB 2600
805 L=L-1:IF L<1 THEN 900
810 J=J+1:IF J>L THEN 870
820 Q=J:GOSUB 2000:GOSUB 1500
830 IF Q=H THEN 810
840 A=J:B=L:GOSUB 2400:L=L-1
850 IF L<1 THEN 900
860 IF L>=J THEN 820
870 RETURN
900 PRINT:PRINT"SOMETHING'S WRONG !!"
910 PRINT:INPUT"WHAT'S YOUR SECRET WORD";P$
915 GOSUB 1800
920 IF F=0 THEN PRINT:PRINT"ILLEGAL WORD";
923 IF F=0 THEN PRINT" - I NEVER HAD A CHANCE"
926 IF F=0 THEN GOTO 1100
930 PRINT
935 PRINT"YOU GAVE A BAD ANSWER SOMEWHERE --"
940 PRINT"CHECK THE SUMMARY":GOSUB 1000
950 GOTO 1100
1000 PRINT:Q=G1:IF G2>G1 THEN Q=G2
1010 IF Q=0 THEN PRINT"NO GUESSES YET":RETURN
1020 FOR J=1 TO 38:PRINT"-";:NEXT:PRINT"-"
1030 PRINT"YOUR GUESSES     ";CHR$(18);"SUMMARY";
1035 PRINT CHR$(146);"       MY GUESSES"
1040 PRINT CHR$(18);"WORD";SPC(2);"HITS";
1045 PRINT TAB(28);"WORD";SPC(2);"HITS"
1050 FOR J=1 TO Q:K=1:IF J>9 THEN K=0
1060 IF J>G1 THEN PRINT SPC(17+K);J;SPC(8);
1063 IF J>G1 THEN PRINT G2$(J);SPC(2);H2(J)
1066 IF J>G1 THEN GOTO 1090
1070 IF J>G2 THEN PRINT SPC(1);G1$(J);SPC(2);
1073 IF J>G2 THEN PRINT H1(J);SPC(8+K);J
1076 IF J>G2 THEN GOTO 1090
1080 PRINT SPC(1);G1$(J);SPC(2);H1(J);SPC(8+K);
1085 PRINT J;SPC(8);G2$(J);SPC(2);H2(J)
1090 NEXT:RETURN
1100 PRINT:INPUT"HOW ABOUT ANOTHER GAME";Q$
1110 Q$=LEFT$(Q$,1):IF Q$="Y" THEN 200
1120 IF Q$="N" THEN END
1130 PRINT:PRINT"YES OR NO PLEASE":GOTO 1100
1500 P$=LEFT$(Q$,1):Q=0:GOSUB 1600
1510 P$=MID$(Q$,2,1):GOSUB 1600
1520 P$=RIGHT$(Q$,1):GOSUB 1600:RETURN
1600 IF P$=M1$ OR P$=M2$ OR P$=M3$ THEN Q=Q+1
1610 RETURN
1800 J=0:F=0
1810 J=J+1:IF J>N THEN RETURN
1820 Q=J:GOSUB 2000:IF Q$<>P$ THEN 1810
1830 F=1:RETURN
```

```
2000 IF Q>V THEN Q$=B$(Q-V):RETURN
2010 Q$=A$(Q):RETURN
2100 IF P>V THEN B$(P-V)=P$:RETURN
2110 A$(P)=P$:RETURN
2200 FOR A=N TO 100 STEP -1:B=INT(RND(1)*A)+1
2210 GOSUB 2400:NEXT
2220 PRINT:PRINT"I'VE ALMOST GOT IT ..."
2230 FOR A=99 TO 2 STEP -1:B=INT(RND(1)*A)+1
2240 GOSUB 2400:NEXT:RETURN
2400 Q=A:GOSUB 2000:P$=Q$:Q=B
2410 GOSUB 2000:P=B:GOSUB 2100:P$=Q$
2420 P=A:GOSUB 2100:RETURN
2600 M1$=LEFT$(Q$,1):M2$=MID$(Q$,2,1)
2610 M3$=RIGHT$(Q$,1):RETURN
3000 RESTORE:FOR P=1 TO N:READ P$
3010 GOSUB 2100:NEXT:RETURN
3200 PRINT:PRINT"IT SURE FEELS GOOD"
3210 PRINT:PRINT"MY WORD WAS - ";M$
3220 GOTO 1100
3400 PRINT
3405 PRINT"CONGRATULATIONS - THAT WAS IT":PRINT
3410 INPUT"WHAT WAS YOUR WORD";P$:GOSUB 1800:J=1
3420 IF F=0 THEN PRINT
3423 IF F=0 THEN PRINT"ILLEGAL WORD";
3426 IF F=0 THEN PRINT" - I HAD NO CHANCE"
3428 IF F=0 THEN GOTO 1100
3430 Q=J:GOSUB 2000
3433 IF Q$=P$ THEN PRINT:PRINT"NICE WORD"
3436 IF Q$=P$ THENGOTO 1100
3440 J=J+1:IF J<=L THEN 3430
3450 PRINT:PRINT"YOU MADE AN ERROR SOMEWHERE"
3455 PRINT"-- CHECK THE SUMMARY"
3460 GOSUB 1000:GOTO 1100
3600 PRINT:PRINT"SORRY, I'M OUT OF MEMORY":PRINT
3610 PRINT"MY WORD WAS - ";M$:GOTO 1100
5000 DATA ACE,ACT,ADE,ADO,ADS,AFT,AGE
5010 DATA AGO,AID,AIL,AIM,AIR,ALE,ALP
5020 DATA AND,ANT,ANY,APE,APT,ARC,ARE
5030 DATA ARK,ARM,ART,ASH,ASK,ASP,ATE
5040 DATA AWE,AWL,AXE,AYE,BAD,BAG,BAN
5050 DATA BAR,BAT,BAY,BED,BEG,BET,BID
5060 DATA BIG,BIN,BIT,BOA,BOG,BOW,BOX
5070 DATA BOY,BUD,BUG,BUM,BUN,BUS,BUT
5080 DATA BUY,BYE,CAB,CAD,CAM,CAN,CAP
5090 DATA CAR,CAT,COB,COD,COG,CON,COP
5100 DATA COT,COW,COY,CRY,CUB,CUD,CUE
5110 DATA CUP,CUR,CUT,DAB,DAM,DAY,DEN
5120 DATA DEW,DIE,DIG,DIM,DIN,DIP,DOE
5130 DATA DOG,DON,DOT,DRY,DUB,DUE,DUG
5140 DATA DYE,DUO,EAR,EAT,EGO,ELK,ELM
```

```
5150 DATA END,EMU,ERA,FAD,FAG,FAN,FAR
5160 DATA FAT,FED,FEW,FIG,FIN,FIR,FIT
5170 DATA FIX,FLY,FOE,FOG,FOR,FOX,FRY
5180 DATA FUN,FUR,GAP,GAS,GAY,GEM,GET
5190 DATA GIN,GNU,GOB,GOD,GOT,FUM,GUN
5200 DATA GUT,GUY,GYP,HAD,HAG,HAM,HAS
5210 DATA HAT,HAY,HEN,HEX,HID,HIM,HIP
5220 DATA HIS,HIT,HER,HEM,HOE,HOG,HOP
5230 DATA HOT,HOW,HUB,HUE,HUG,HUM,HUT
5240 DATA ICE,ICY,ILK,INK,IMP,ION,IRE
5250 DATA IRK,ITS,IVY,JAB,JAR,JAW,JAY
5260 DATA JOB,JOG,JOT,JOY,JUG,JAG,JAM
5270 DATA JET,JIB,JOG,JUT,KEG,KEY,KID
5280 DATA KIN,KIT,LAB,LAD,LAG,LAP,LAW
5290 DATA LAY,LAX,LED,LEG,LET,LID,LIE
5300 DATA LIP,LIT,LOB,LOG,LOP,LOT,LOW
5310 DATA LYE,MAD,MAN,MAP,MAR,MAT,MAY
5320 DATA MEN,MET,MID,MOB,MOP,MOW,MUD
5330 DATA MIX,MUG,NAB,NAG,NAP,NAY,NET
5340 DATA NEW,NIL,NIP,NOD,NOT,NOR,NOW
5350 DATA NUT,OAF,OAK,OAR,OAT,ODE,OIL
5360 DATA OLD,ONE,OPT,ORE,OUR,OUT,OVA
5370 DATA OWE,OWL,OWN,PAD,PAL,PAN,PAR
5380 DATA PAT,PAW,PAY,PEA,PEG,PEN,PET
5390 DATA PEW,PIE,PIG,PIT,PLY,POD,POT
5400 DATA POX,PER,PIN,PRO,PRY,PUB,PUN
5410 DATA PUS,PUT,RAG,RAM,RAN,RAP,RAT
5420 DATA RAW,RAY,RED,RIB,RID,REV,RIG
5430 DATA RIM,RIP,ROB,ROD,ROE,ROT,ROW
5440 DATA RUB,RUE,RUG,RUM,RUN,RUT,RYE
5450 DATA SAD,SAG,SAP,SAT,SAW,SAY,SET
5460 DATA SEW,SEX,SHY,SEA,SIN,SHE,SIP
5470 DATA SIR,SIT,SIX,SKI,SKY,SLY,SOB
5480 DATA SOD,SON,SOW,SOY,SPA,SPY,STY
5490 DATA SUE,SUM,SUN,TAB,TAD,TAG,TAN
5500 DATA TAP,TAX,TAR,TEA,TEN,THE,THY
5510 DATA TIC,TIE,TIN,TIP,TOE,TON,TOP
5520 DATA TOW,TOY,TRY,TUB,TUG,TWO,URN
5530 DATA USE,UPS,VAN,VAT,VEX,VIA,VIE
5540 DATA VIM,VOW,YAK,YAM,YEN,YES,YET
5550 DATA YOU,WAD,WAG,WAN,WAR,WAS,WAX
5560 DATA WAY,WEB,WED,WET,WHO,WHY,WIG
5570 DATA WIN,WIT,WOE,WON,WRY,ZIP,FIB

READY.
```

EASY CHANGES

1. It is fairly common for players to request a summary before most guesses that they make. If you want the program to

automatically provide a summary before each guess, change line 500 to read

```
500 GOSUB 1000:PRINT:INPUT"YOUR GUESS
(OR Q)";P$
```

and delete line 505.

2. The maximum number of guesses allowed, M, can be changed in line 150. You may wish to increase it in conjunction with Suggested Project 2. You might decrease it to free some memory needed for other program additions. The current value of twenty-five is really somewhat larger than necessary. An actual game almost never goes beyond fifteen guesses. To set M to 15 change line 150 to read

```
150 M=15:N=406:V=250
```

3. Modifying the data list of legal words is fairly easy. Our criteria for legal words were as follows: they must have three distinct letters and *not* be

   - capitalized
   - abbreviations
   - interjections (like "ugh," "hey" etc.)
   - specialized words (like "ohm," "sac," "yaw" etc.)

   In line 150, N is set to be the total number of words in the data list. The data list itself is from line 5000 on.

   To add word(s), do the following. Enter them in data statements after the current data (use line numbers larger than 5570). Then redefine the value of N to be 406 plus the number of new words added. For example, to add the words "ohm" and "yaw" onto the list, change line 150 to read

```
150 M=25:N=408:V=250
```

   and add a new line

```
5580 DATA OHM,YAW
```

   To delete word(s), the opposite must be done. Remove the words from the appropriate data statement(s) and decrease the value of N accordingly.

MAIN ROUTINES

150- 170	Dimensions arrays.
200- 330	Initializes new game.

500- 570	Human guesses at the computer's word.
600- 680	Computer guesses.
800- 870	Evaluates human's possible secret words. Moves them to the front of A$-B$ array.
900- 950	Processes inconsistency in given information.
1000-1090	Displays the current summary table.
1100-1130	Inquires about another game.
1500-1610	Compares a guess with key word.
1800-1830	Checks if input word is legal.
2000-2010	Sets Q$ to Qth element of A$-B$ array.
2100-2110	Sets Pth element of A$-B$ array to P$.
2200-2240	Shuffles A$-B$ array randomly.
2400-2420	Swaps elements A and B in the A$-B$ array.
2600-2610	Breaks word Q$ into separate letters.
3000-3010	Fills A$-B$ array from data.
3200-3220	Post-mortem after computer wins.
3400-3460	Post-mortem after human wins.
3600-3610	Error routine—too many guesses.
5000-5520	Data.

MAIN VARIABLES

V	Size of A$ array (250).
N	Total number of data words.
M	Maximum number of guesses allowed.
A$	String array holding first V data words.
B$	String array holding last (N-V) words.
G1$,G2$	String arrays of human's, computer's guesses.
H1,H2	Arrays of human's, computer's hits corresponding to G1$,G2$.
G1,G2	Current number of human's, computer's guesses.
M$	Computer's secret word.
M1$,M2$, M3$	First, second, and third letters of a word.
P$,Q$	String temporaries and work variables.
L	Current number of human's possible secret words.
F	Flag for input word legality.
H	Number of hits in last guess.
A,B	A$-B$ array locations to be swapped.
J,P,Q	Temporaries; array and loop indices.
K	Formatting variable used in summary display.

SUGGESTED PROJECTS

1. Additional messages during the course of the game can personify the program even more. After the Commodore 64 finds out how its last guess did, you might try an occasional message like one of these:

   JUST AS I THOUGHT...
   HMM, I DIDN'T EXPECT THAT...
   JUST WHAT I WAS HOPING TO HEAR...

   The value of L is the number of words to which the computer has narrowed down the human's secret word. You might check its value regularly and when it gets low, come out with something like

   BE CAREFUL, I'M CLOSING IN ON YOU.

2. Incorporate a feature to allow the loser to continue guessing at the other's word. The summary display routine will already work fine even if G1 and G2 are very different from each other. It will display a value of "9" for the number of hits corresponding to the correct guess of a secret word.

OBSTACLE

PURPOSE

This program allows you and a friend (or enemy) to play the game of OBSTACLE, an arcade-like game that's one of our favorites. A combination of physical skills (reflex speed, hand to eye coordination, etc.) and strategic skills are needed to beat your opponent. Each game generally takes only a minute or two, so you'll want to play a match of several games to determine the better player.

HOW TO USE IT

The object of the game is to keep moving longer than your opponent without bumping into an obstacle. When the program starts, it asks in turn for the name of the player on the left and on the right. Then it displays the playing field, shows the starting point for each player, and tells you to press any key to start.

After a key is pressed, each player begins moving independently in one of four random directions—up, down, left, or right. As each player moves, he or she builds a "wall" inside the playing field. The computer determines the speed of the move; the player can only control his own direction. The player on the left can change direction to up, down, left, or right by pressing the key **W**, **Z**, **A**, or **D**, respectively. The player on the right does the same by using the keys for @, / (slash), : (colon), and = (equals). Find these keys on the Commodore 64 keyboard and you will see the logic behind these choices.

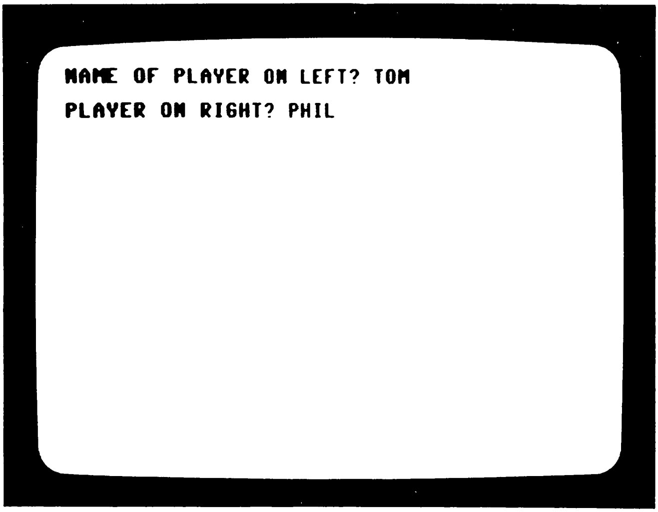
The first time either player bumps into the wall surrounding the playing field or the obstacle wall built by either player, he or

she loses. When this happens, the program indicates the point of impact for a few seconds and displays the name of the winner. Then the game starts over.

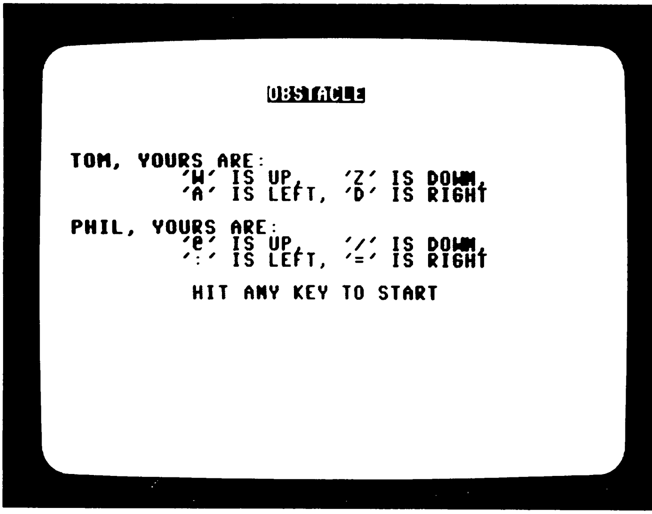
The strategic considerations for this game are interesting. Should you attack your opponent, trying to build a wall around him that he must crash into? Or should you stay away from him and try to make efficient moves in an open area until your opponent runs out of room on his own? Try both approaches and see which yields the most success.

When pressing a key to change direction, be sure to press it quickly and release it. *Do not* hold a key down—you might inhibit the computer from recognizing a move your opponent is trying to make. Once in a while, only one key will be recognized when two are hit at once.

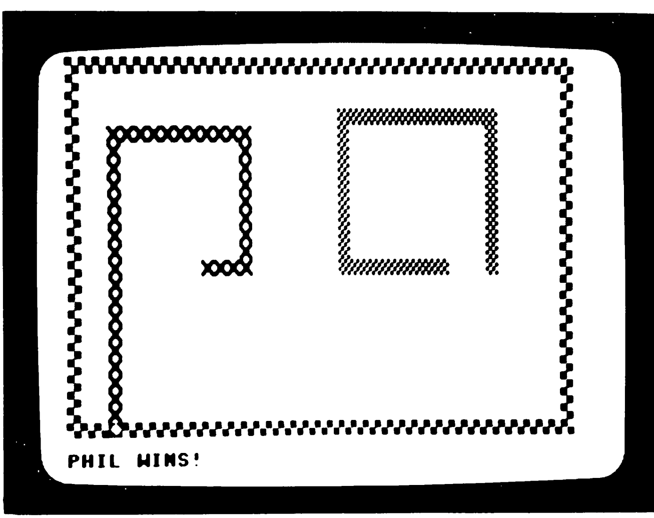
SAMPLE RUN



The program starts off by asking for the names of the two players.



The program tells each player which keys to use to change directions and waits for a key to be pressed.



The program draws the playing field and starts each player moving in a random direction. Tom (on the left) doesn't change direction soon enough and crashes into the wall, making Phil the winner.

PROGRAM LISTING

```
READY.

100 REM: OBSTACLE
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
115 PRINT CHR$(5)
120 X=RND(-TI):GOSUB 600
125 POKE 53281,1:PRINT CHR$(147):PRINT
128 POKE 53280,1:POKE 53281,0
130 PRINT TAB(15);CHR$(18);"OBSTACLE"
140 PRINT:PRINT
141 PRINT:PRINT A$;", YOURS ARE:"
142 PRINT TAB(8) "'W' IS UP,   'Z' IS DOWN,"
143 PRINT TAB(8) "'A' IS LEFT, 'D' IS RIGHT"
145 PRINT:PRINT B$;", YOURS ARE:"
146 PRINT TAB(8) "'@' IS UP,   '/' IS DOWN,"
147 PRINT TAB(8) "':' IS LEFT, '=' IS RIGHT"
150 PRINT:PRINT TAB(9);"HIT ANY KEY TO START"
152 GET R$:IF R$="" THEN 152
155 Z=90
160 AX=10:AY=12:BX=29:BY=12:A=86:B=102
165 S=1024:E=127:AD=INT(4*RND(1))+1
167 BD=INT(4*RND(1))+1:CO=55296:AC=2:BC=5
170 POKE 53281,1:PRINT CHR$(147);:POKE 53281,0
190 FOR J=1 TO 1000:NEXT
200 GOSUB 950:GOSUB 900
205 FOR J=1 TO 10:GET R$:NEXT
210 X=AX:Y=AY:D=AD:GOSUB 1000
220 AR=R:AX=X:AY=Y
230 X=BX:Y=BY:D=BD:GOSUD 1000
240 BR=R:BX=X:BY=Y
245 IF AR=1 OR BR=1 THEN 400
250 GOSUB 900
255 FOR J=1 TO 10
260 GET R$
265 IF R$="W" THEN AD=1
270 IF R$="Z" THEN AD=2
280 IF R$="A" THEN AD=3
290 IF R$="D" THEN AD=4
300 IF R$="@" THEN BD=1
310 IF R$="/" THEN BD=2
320 IF R$=":" THEN BD=3
330 IF R$="=" THEN BD=4
340 NEXT
350 GOTO 210
400 GOSUB 700:X=AX:Y=AY
410 IF BR=1 THEN X=BX:Y=BY
```

```
420 FOR J=1 TO 15
430 POKE S+40*Y+X,Z
440 FOR K=1 TO 200:NEXT
450 POKE S+40*Y+X,Z+128
460 FOR K=1 TO 200:NEXT
470 NEXT
480 FOR J=1 TO 20:GET R$:NEXT
490 GOTO 125
600 POKE 53281,1:PRINT CHR$(147):POKE 53281,0
610 INPUT"NAME OF PLAYER ON LEFT";A$
620 PRINT
630 INPUT"PLAYER ON RIGHT";B$
640 RETURN
700 PRINT CHR$(19);
710 FOR J=1 TO 12
720 PRINT CHR$(17):NEXT
730 IF AR=1 AND BR=1 THEN PRINT"YOU BOTH LOSE!";
735 IF AR=1 AND BR=1 THEN RETURN
740 R$=A$:IF AR=1 THEN R$=B$
750 PRINT R$;" WINS!";
760 RETURN
900 POKE (CO+40*AY+AX),AC
905 POKE (S+40*AY+AX),A
910 POKE (CO+40*BY+BX),BC
915 POKE (S+40*BY+BX),B
920 RETURN
950 FOR X=0 TO 39
960 POKE S+X,E:POKE S+880+X,E
970 NEXT:FOR Y=0 TO 22
980 POKE (S+40*Y),E:POKE (S+40*Y+39),E:NEXT
990 RETURN
1000 IF D=1 THEN Y=Y-1
1010 IF D=2 THEN Y=Y+1
1020 IF D=3 THEN X=X-1
1030 IF D=4 THEN X=X+1
1040 R=0
1050 IF PEEK(S+40*Y+X)<>32 THEN R=1
1060 RETURN

READY.
```

EASY CHANGES

1. To speed the game up, change the 10 in line 255 to a 5 or so. To slow it down, make it 12 or 15.
2. To make both players always start moving upward at the beginning of each game (instead of in a random direction), insert the following statement:

   168 AD=1:BD=1

To make the players always start off moving toward each other, use this statement instead:

168 AD=4:BD=3

3. To change the length of time that the final messages are displayed after each game, modify line 420. Change the 15 to 8 (or so) to shorten it, or to 25 to lengthen it.
4. Change the keys that are used to determine each player's direction by altering the appropriate values in lines 265 through 330. For example, to make the X key cause the player on the left to go down, make this change:

270 IF R$="X" THEN AD=2

Don't forget to change the display in lines 141 through 147 to agree.

MAIN ROUTINES

120- 150	Initializes variables. Gets players' names. Displays titles, playing field.
152- 200	Waits for key to be pressed to start game. Redisplays playing field.
205- 250	Makes move for player A (on left side) and B (on right). Saves results.
255- 350	Accepts moves from keyboard and translates direction.
400- 490	Displays winner's name at bottom of screen. Flashes a diamond where collision occurred. Goes back to start next game.
600- 640	Subroutine that gets each player's name.
700- 760	Subroutine that displays winner's name.
900- 920	Subroutine that POKEs each graphics character of each player's obstacle on the screen.
950- 990	Subroutine that displays playing field.
1000-1060	Subroutine that moves marker and determines if space moved to is empty.

MAIN VARIABLES

AX,AY	Player A's current coordinates.
BX,BY	Player B's current coordinates.
A	A's marker (numeric value of graphics character).
B	B's marker.
S	Starting address of CRT memory area.

AD,BD	Current direction that A and B are going (1 = up, 2 = down, 3 = left, 4 = right).
E	Graphics character for edge of playing field.
R$	Character being read from keyboard; also work variable.
CO	Starting address of CRT color map.
AC	Color of A's marker.
BC	Color of B's marker.
X,Y	Temporary work coordinates.
AR,BR	Result of A's and B's moves (0 = okay, 1 = loser).
A$,B$	Names of players A and B.
Z	Graphics character for collision.
J,K	Subscript variables.

SUGGESTED PROJECTS

1. Keep score over a seven game (or so) match. Display the current score after each game. Don't forget to allow for ties.
2. Modify the program to let each player press only two keys—one to turn left from the current direction of travel, and one to turn right.
3. Instead of a game between two people, make it a game of a person against the computer. Develop a computer strategy to keep finding open areas to move to and/or to cut off open areas from the human opponent.
4. Change the size of the playing field. The 39 in line 950 is the width and the 22 in line 970 is the height. Note that line 160 has the starting coordinates of the two players (AX, AY, BX, and BY). You may want to change these if you make the field smaller.

ROADRACE

PURPOSE

Imagine yourself at the wheel of a high-speed race car winding your way along a treacherous course. The road curves unpredictably. To stay on course, you must steer accurately or risk collision. How far can you go in one day? How many days will it take you to race cross-country? Thrills galore without leaving your living room.

The difficulty of the game is completely under your control. By adjusting the road width and visibility conditions, ROADRACE can be made as easy or as challenging as you wish.

HOW TO USE IT

The program begins with a short graphics display. It then asks you to hit any key to begin. Next you are requested to provide two inputs: road width and visibility. The road width (in characters) can be set anywhere between 3 and 12. The degree of difficulty changes appreciably with different widths. A very narrow setting will be quite difficult and a wide one relatively easy. Visibility can be set to any of four settings, ranging from "terrible" to "good." When visibility is good, the car appears high on the screen. This allows a good view of the twisting road ahead. When visibility is poor, the car appears low on the screen allowing only a brief look at the upcoming road.

Having set road width and visibility, the race is ready to start. The car appears on the road at the starting line. A five-step starting light counts down the start. When the bottom light goes on,

the race begins. The road moves continually up the screen. Its twists and turns are controlled randomly. You must steer the car accurately to keep it on track.

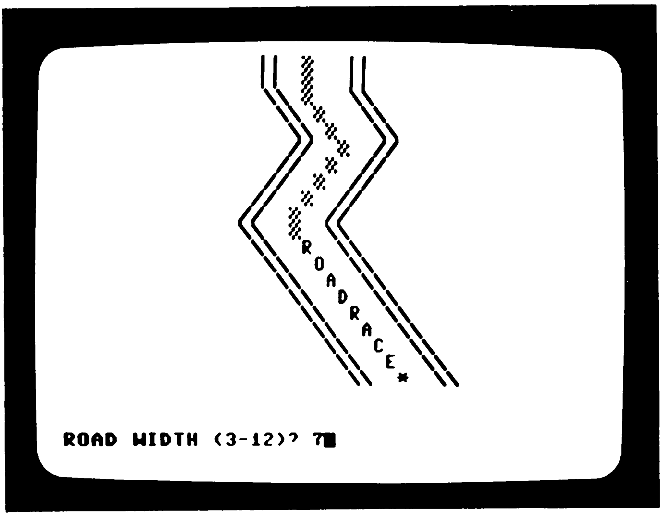
The car is controlled with the use of two keys near the lower left and right corners of the keyboard. Pressing the **Z** will cause the car to move to the left while pressing the / (slash) will cause a move to the right. Doing neither will cause the car to continue straight down.

The race proceeds until the car goes "off the road." Each such collision is considered to terminate one day of the race. After each day, you are shown the number of miles achieved that day along with the cumulative miles achieved for consecutive days of the race.

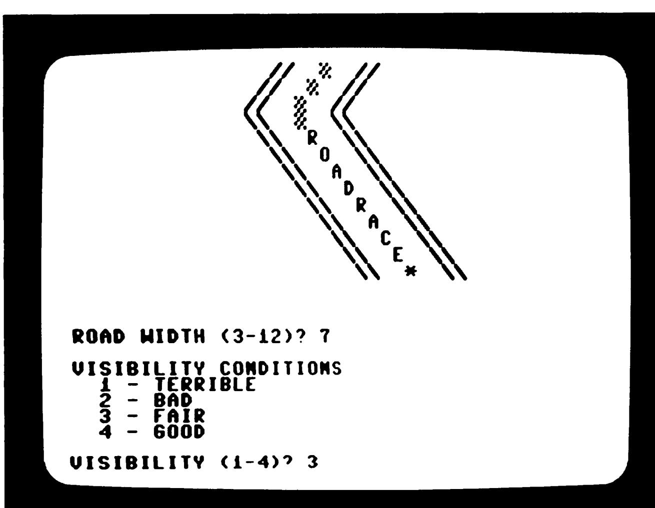
After each collision, you can proceed by pressing either **C**, **R**, or **Q**. Selecting **C** will continue the race for another day with the same road conditions. Cumulative totals will be retained. **R** will restart the race. This allows changing the road conditions and initializing back to day one. **Q** simply quits the race and returns the computer back to direct Basic. Either of the last two options will produce a display of the average miles travelled per day for the race.

There are several different ways to challenge yourself with the program. You can try to see how far you get in a given number of days. You might see how many days it takes you to go a given number of miles—say 3000 miles for a cross-country trip. As you become proficient at one set of road conditions, make the road narrower and/or the visibility poorer. This will increase the challenge. Different road conditions can also be used as a handicapping aid for two unequally matched opponents.

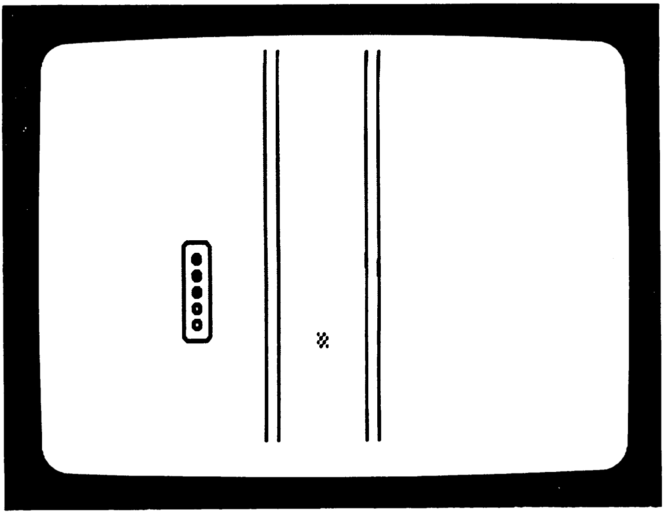
SAMPLE RUN



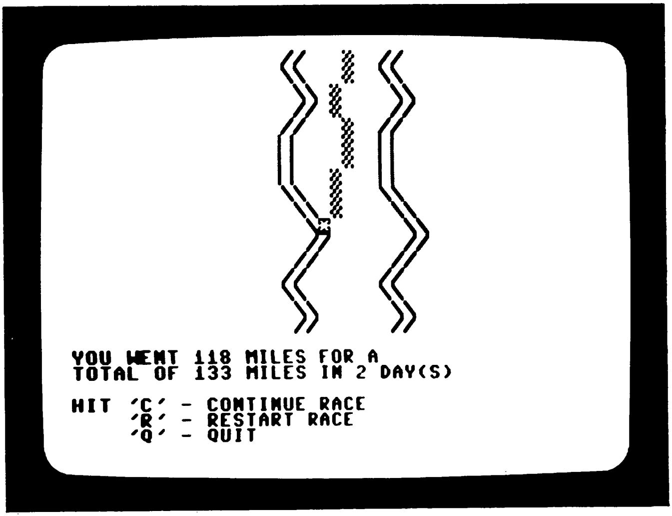
The program displays the logo and begins the short input phase. The operator selects a course with a 7 character road width.



The operator selects a course with fair visibility.



The car is on the starting line. The starting light counts down the beginning of the race. When the last light goes on, the race will be off and running.



The operator, steering the car from the keyboard, finally crashes. A distance of 118 miles is travelled on this leg for a total of 133 miles in 2 days. The options for continuing are displayed while the program waits for the operator's choice.

PROGRAM LISTING

```
READY.

100 REM: ROADRACE
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
120 Q=0:Q$=""
125 OS=54272
130 LC=.4:RC=1-LC
140 RS=167:LS=165:LT=206:RT=205
150 L$="Z":R$="/"
160 B=32:PC=102:EL=2:ER=37
165 POKE 53280,8:POKE 53281,2:PRINT CHR$(5)
170 GOSUB 800
200 PRINT:PRINT:T=0:N=0
210 INPUT"ROAD WIDTH (3-12)";W
220 W=INT(W):IF W<3 OR W>12 THEN 210
230 PRINT:PRINT"VISIBILITY CONDITIONS"
240 PRINT"  1 - TERRIBLE"
250 PRINT"  2 - BAD"
260 PRINT"  3 - FAIR"
270 PRINT"  4 - GOOD":PRINT
280 INPUT"VISIBILITY (1-4)";V:V=INT(V)
290 IF V<1 OR V>4 THEN 280
295 POKE 53280,8:POKE 53281,2
300 N=N+1:L=14:R=L+W+2:Z=2064-120*V
310 C=INT((L+R)/2)+1
320 FOR J=1 TO 26:GET Q$:GOSUB 600:NEXT
330 Q=RND(-TI):GOSUB 700
350 Q=RND(1)
355 IF Q>RC AND R<ER THEN GOSUB 640:GOTO 400
360 IF Q<LC AND L>EL THEN GOSUB 620:GOTO 400
370 GOSUB 600
400 A=PC:GET Q$:IF Q$=L$ THEN C=C-1
410 ZC=Z+C:IF Q$=R$ THEN C=C+1
420 Q=PEEK(ZC):IF Q<>B THEN A=170
430 POKE ZC,A:POKE ZC+OS,1:IF A=PC THEN 350
440 H=TI-H:IF H<0 THEN H=H+5184000
450 M=INT(H/10):T=T+M:PRINT
460 PRINT"YOU WENT";M"MILES FOR A"
470 PRINT"TOTAL OF";T;"MILES IN";N;"DAY(S)":PRINT
480 PRINT"HIT 'C' - CONTINUE RACE"
490 PRINT"     'R' - RESTART RACE"
500 PRINT"     'Q' - QUIT
510 GET Q$:IF Q$="C" THEN 300
520 IF Q$<>"R" AND Q$<>"Q" THEN 510
530 PRINT:PRINT"AVERAGE MILES PER DAY=";T/N
540 IF Q$="R" THEN 200
550 END
600 PRINT TAB(L);CHR$(RS);CHR$(RS);
```

```
605 PRINT TAB(R);CHR$(LS);CHR$(LS)
610 RETURN
620 PRINT TAB(L);CHR$(LT);CHR$(LT);
625 PRINT TAB(R-1);CHR$(LT);CHR$(LT)
630 L=L-1:R=R-1:RETURN
640 PRINT TAB(L+1);CHR$(RT);CHR$(RT);
645 PRINT TAB(R);CHR$(RT);CHR$(RT)
650 L=L+1:R=R+1:RETURN
700 POKE Z+C,PC:POKE Z+C+OS,1:P=Z-232
710 POKE P,85:POKE P+OS,1:POKE P+1,64
720 POKE P+1+OS,1:POKE P+2,73:POKE P+2+OS,1
730 FOR J=1 TO 5:POKE P+40*J,93:POKE P+40*J+OS,1
740 POKE P+40*J+1,87:POKE P+40*J+1+OS,1
750 POKE P+40*J+2,93:POKE P+40*J+2+OS,1
760 NEXT:POKE P+240,74:POKE P+240+OS,1
770 POKE P+241,64:POKE P+241+OS,1:POKE P+242,75
780 POKE P+242+OS,1:FOR J=1 TO 900:NEXT
790 FOR J=1 TO 5:FOR K=1 TO 400:NEXT
795 POKE P+40*J+1,81:NEXT:H=TI:RETURN
800 DIM D(9):L=14:R=22:POKE 53281,1
805 PRINT CHR$(147);:POKE 53281,2
810 FOR J=1 TO 9:READ D(J):NEXT
820 DATA 18,15,1,4,18,1,3,5,42
830 FOR J=1 TO 2:GOSUB 600:NEXT
840 FOR J=1 TO 3:GOSUB 640:NEXT
850 FOR J=1 TO 5:GOSUB 620:NEXT
860 FOR J=1 TO 10:GOSUB 640:NEXT
870 P=1042:POKE P,PC
880 FOR J=1 TO 500:NEXT
890 FOR J=1 TO 2:P=P+40:GOSUB 950:NEXT
900 FOR J=1 TO 3:P=P+41:GOSUB 950:NEXT
910 FOR J=1 TO 4:P=P+39:GOSUB 950:NEXT
920 P=P+40:GOSUB 950
930 FOR J=1 TO 9:P=P+41:GOSUB 950
940 POKE P,D(J):NEXT:PRINT:RETURN
950 POKE P,PC:FOR K=1 TO 50:NEXT:RETURN

READY.
```

EASY CHANGES

1. The keys which cause the car to move left and right can be easily changed. You may wish to do this if you find that two widely separated keys are less convenient. The changes are to be made in line 150. Left and right movements are controlled by the two string variables L$ and R$. If, for example, you wanted **1** to cause a left move and **9** to cause a right move, change line 150 to read

   150 L$ = "1":R$ = "9"

2. The amount of windiness in the road can be adjusted by changing the value of LC in line 130. Maximum windiness is achieved with a value of 0.5 for LC. To get a straighter road, make LC smaller. A value of 0. will produce a completely straight road. LC should lie between 0. and 0.5 or else the road will drift to one side and linger there. To get a somewhat less winding road, you might change line 130 to read

```
130 LC=0.3:RC=1-LC
```

MAIN ROUTINES

120-170	Variable initialization.
200-290	Gets road conditions from user.
300-330	Initializes the road.
350-370	Determines the next road condition.
400-430	Updates the car position.
440-450	Processes end of race day.
600-650	Draws next road segment.
700-795	Graphics to begin race.
800-950	Initial graphics display.

MAIN VARIABLES

W	Road width.
V	Visibility.
M	Miles driven on current day.
N	Number of days of the race.
T	Total miles driven for whole race.
H	Elapsed time during race.
L$,R$	String characters to move car left, right.
L,R	Position of left and right sides of road.
LC,RC	Random value cutoff to move road left, right.
EL,ER	Leftmost, rightmost allowable road position.
Q$	User replies.
C	Screen location of car.
RS,LS, LT,RT	Arguments of CHR$ for road segments.
B,PC	POKE arguments for graphics.
A,P	POKE arguments.
D	Array of POKE arguments for display message.

J,K,Q	Loop indices and work variables.
Z,ZC	POKE arguments for car location.

SUGGESTED PROJECTS

1. Write a routine to evaluate a player's performance after each collision. Display a message rating him anywhere from "expert" to "back seat driver." This should involve comparing his actual miles achieved against an expected (or average) number of miles for the given road width and visibility. For starters, you might use

   $$\text{Expected miles} = W^3 + (10*V) - 35$$

   This formula is crude, at best. The coding can be done between lines 550 and 600.
2. Incorporate provisions for two players racing one at a time. Keep cumulative totals separately. After each collision, display the current leader and how far he is ahead.
3. Add physical obstacles or other hazards onto the road in order to increase the challenge. This can be done with appropriate POKE statements before the various RETURNs in lines 600-650. The program will recognize a collision if the car moves into any non-blank square. WARNING—Be sure the address arguments of any POKEs lie between 1024 and 2023. Anything else may result in hanging up your system or other strange occurrences.

WARI

PURPOSE

Wari is an old game with roots that are much older. Its origins go back thousands of years to a variety of other similar games, all classified as being members of the Mancala family. Other variations are Awari, Oware, Pallanguli, Kalah, and countless other offshots.

The program matches you against the computer. You are probably going to lose a few games before you win one—the computer plays a pretty good game. This may hurt your ego a little bit, since Wari is purely a skill game (like chess or checkers). There is no element of luck involved, as would be the case with backgammon, for example. When you lose, it's because you were outplayed.

HOW TO USE IT

When you start the program, the first thing it does is display the Wari board and ask you if you want to go first. The board is made up of twelve squares in two rows of six. Your side is the bottom side, numbered one through six from left to right. The computer's side is on the top, numbered seven through twelve from right to left.

At the start of the game, each square has four "stones" in it. There is no way to differentiate between your stones and the computer's. They all look alike and will move from one side to the other during the course of play.

The first player "picks up" all the stones in one of the squares on his side of the board and drops them, one to a square, start-

ing with the next highest numbered square. The stones continue to be dropped consecutively in each square, continuing over onto the opponent's side if necessary (after square number 12 comes square number 1 again).

If the last stone is dropped onto the opponent's side *and* leaves a total of either two or three stones in that square, these stones are captured by the player who moved, and removed from the board. Also, if the next-to-last square in which a stone was dropped meets the same conditions (on the opponent's side and now with two or three stones), its stones are also captured. This continues backwards until the string of consecutive squares of two or three on the opponent's side is broken.

Regardless of whether any captures are made, play alternates back and forth between the two players.

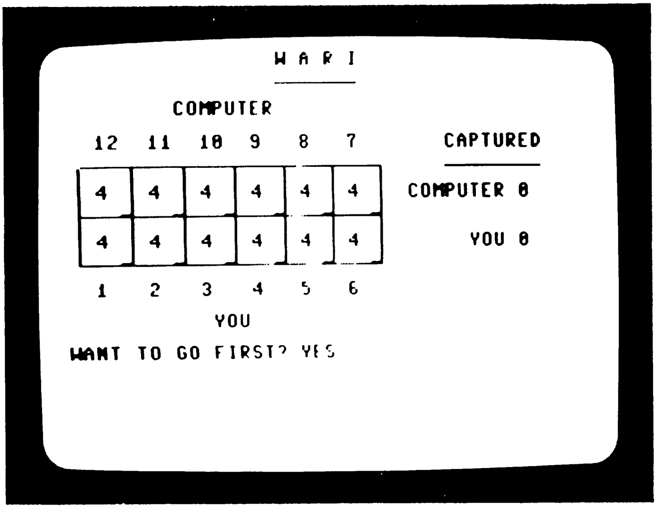
The object of the game is to be the first player to capture twenty-four or more stones. That's half of the forty-eight stones that are on the board at the beginning of the game.

There are a few special rules to cover some situations that can come up in the game. It is not legal to capture all the stones on the opponent's side of the board, since this would leave the opponent with no moves on his next turn. By the same token, when your opponent has no stones on his side (because he had to move his last one to your side on his turn), you have to make a move that gives him at least one stone to move on his next turn, if possible. If you cannot make such a move, the game is over and counted as a draw.

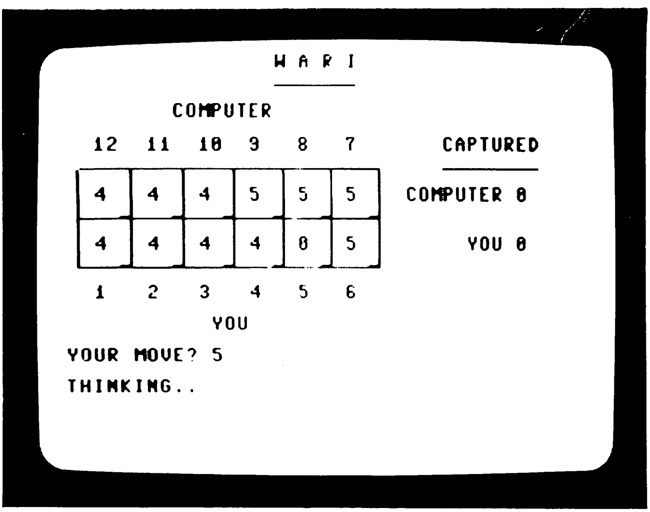
During the course of the game, it's possible for a square to accumulate twelve or more stones in it. Moving from such a square causes stones to be distributed all the way around the board. When this happens, the square from which the move was made is skipped over. So, the square moved from is always left empty.

It takes the computer anywhere from five seconds to about forty seconds to make a move, depending on the complexity of the board position. The word THINKING is displayed during this time, and a period is added to it as each possible move is evaluated in sequence (seven through twelve).

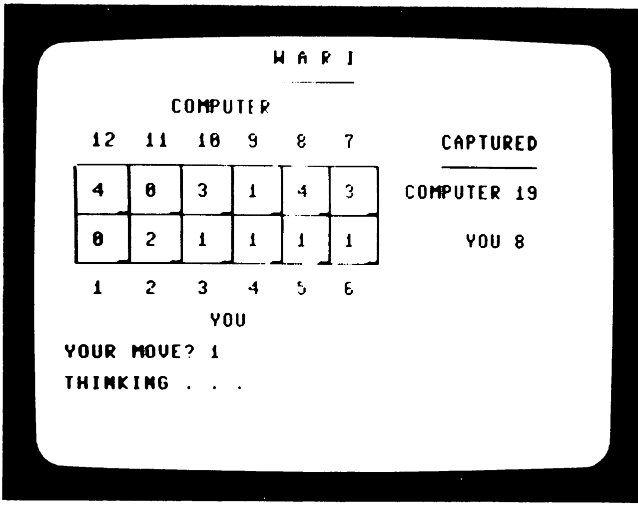
SAMPLE RUN



The program starts off by drawing the playing board and asking who should move first. The operator decides to go first.



The program asks for the operator's move. He or she decides to move square number 5. The program alters the board accordingly, and begins "thinking" about what move to make.



Later in the same game, the computer is about to move square number 12, which will capture 7 more stones and win the game.

PROGRAM LISTING

```
READY.

100 REM: WARI
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 J=1:K=1:Q=14:P=13:F=50:D=12
125 POKE 53280,2:POKE 53281,9:PRINT CHR$(5)
130 DIM T(Q),Y(Q),W(Q),V(6),E(6),B(Q)
140 WA=RND(-TI):WB=RND(1):WB=WB/Q
145 WA=.25+WB:WB=.25-WB:GOSUB 750
150 FOR J=1 TO D:B(J)=4:NEXT
155 B(P)=0:B(Q)=0:MN=0:GOSUB 1200:GOSUB 900
160 GOSUB 990:INPUT"WANT TO GO FIRST";R$
170 GOSUB 990:PRINT D$:R$=LEFT$(R$,1)
175 IF R$="Y" THEN 250
180 IF R$<>"N" THEN 160
190 GOSUB 1050:PRINT D$;D$;D$:GOSUB 1050
192 PRINT"THINKING";:GOSUB 510
```

```
195 IF M<1 THEN 2000
200 GOSUB 1050:PRINT D$:GOSUB 1050
205 PRINT"MY MOVE IS";M
210 FOR J=1 TO Q:T(J)=B(J):NEXT:GOSUB 350
220 FOR J=1 TO Q:B(J)=T(J):NEXT:GOSUB 900
230 IF B(Q)<24 THEN 250
240 GOSUB 1050:PRINT"I WIN!";D$:GOTO 810
250 GOSUB 990:PRINT D$;D$:GOSUB 990
255 INPUT"YOUR MOVE";R$
260 M=INT(VAL(R$)):IF M>6 OR M<1 THEN 330
270 FOR J=1 TO Q:T(J)=B(J):NEXT
280 GOSUB 350:IF M<0 THEN 330
290 FOR J=1 TO Q:B(J)=T(J):NEXT
300 MN=MN+1:GOSUB 900
310 PRINT:IF B(P)<24 THEN 190
320 GOSUB 1050:PRINT"YOU WIN!";D$:GOTO 810
330 GOSUB 990:PRINT TAB(15);CHR$(18);"ILLEGAL"
340 FOR J=1 TO 3000:NEXT:GOTO 250
350 IF T(M)=0 THEN M=-1:RETURN
360 R$="H":IF M>6 THEN R$="C":GOTO 380
370 FOR J=1 TO Q:Y(J)=T(J):NEXT:GOTO 400
380 FOR J=1 TO 6:Y(J)=T(J+6):Y(J+6)=T(J):NEXT
390 Y(P)=T(Q):Y(Q)=T(P):M=M-6
400 C=M:N=Y(C):FOR J=1 TO N:C=C+1
410 IF C=P THEN C=1
420 IF C=M THEN C=C+1:GOTO 410
430 Y(C)=Y(C)+1:NEXT:Y(M)=0:L=C
440 IF L<7 OR Y(L)>3 OR Y(L)<2 THEN 460
450 Y(P)=Y(P)+Y(L):Y(L)=0:L=L-1:GOTO 440
460 S=0:FOR J=7 TO D:S=S+Y(J):NEXT
470 IF S=0 THEN M=-2:RETURN
480 IF R$="H" THEN FOR J=1 TO Q:T(J)=Y(J):NEXT
485 IF R$="H" THEN RETURN
490 FOR J=1 TO 6:T(J)=Y(J+6):T(J+6)=Y(J):NEXT
500 T(Q)=Y(P):T(P)=Y(Q):RETURN
510 FOR A=1 TO 6:M=A+6
520 IF B(M)=0 THEN E(A)=-F:GOTO 690
530 FOR J=1 TO Q:T(J)=B(J):NEXT:GOSUB 350
540 IF M<0 THEN E(A)=-F:GOTO 690
550 IF T(Q)>23 THEN M=A+6:RETURN
560 FOR J=1 TO Q:W(J)=T(J):NEXT:FOR K=1 TO 6
570 IF T(K)=0 THEN V(K)=F:GOTO 670
580 FOR J=1 TO Q:T(J)=W(J):NEXT:M=K:GOSUB 350
590 IF M<0 THEN V(K)=F:GOTO 670
600 FA=0:FB=.05:FC=0:FD=0:FOR J=7 TO D
610 FB=FB+T(J):IF T(J)>0 THEN FA=FA+1
620 IF T(J)<3 THEN FC=FC+1
630 IF T(J)>FD THEN FD=T(J)
640 NEXT:FE=FB:FOR J=1 TO 6:FE=FE+T(J):NEXT
650 FA=FA/6:FD=1-FD/FB:FC=1-FC/6:FB=FB/FE
```

```
660 V(K)=WA*(FA+FB)+WB*(FC+FD)+T(Q)+B(P)-B(Q)-T(P)
670 NEXT:E(A)=F
675 FOR J=1 TO 6:IF V(J)<E(A) THEN E(A)=V(J)
680 NEXT
690 PRINT".";:NEXT:M=0:A=-F:FOR J=1 TO 6
700 IF E(J)>A THEN A=E(J):M=J+6
710 NEXT:RETURN
750 A$=" ":FOR J=1 TO 24:A$=A$+CHR$(164):NEXT
760 B$=CHR$(167):FOR J=1 TO 6
765 B$=B$+"   "+CHR$(167):NEXT
770 C$=CHR$(167):FOR J=1 TO 6
780 C$=C$+CHR$(164)+CHR$(164)+CHR$(164)+CHR$(186)
790 NEXT:D$=" ":FOR J=1 TO 5:D$=D$+D$:NEXT
800 RETURN
810 PRINT:J=ABS(B(P)-B(Q)):IF J<10 THEN 830
820 PRINT"IT WASN'T EVEN CLOSE!":GOTO 840
830 PRINT"GOOD GAME!"
840 PRINT:INPUT"WANT TO PLAY AGAIN";R$
850 R$=LEFT$(R$,1):IF R$="Y" THEN 140
860 IF R$<>"N" THEN 840
870 PRINT:PRINT:PRINT"SEE YOU LATER"
880 PRINT:PRINT:END
900 PRINT CHR$(19);
910 FOR J=1 TO 4:PRINT CHR$(17):NEXT
920 FOR J=0 TO 5:PRINT TAB(4*J+1);B(12-J);
925 IF B(12-J)=0 THEN GOSUB 1100
930 NEXT:PRINT TAB(27);"COMPUTER";B(Q)
940 PRINT CHR$(17):FOR J=0 TO 5
950 PRINT TAB(4*J+1);B(J+1);
955 IF B(J+1)=0 THEN GOSUB 1100
960 NEXT:PRINT TAB(32);"YOU";B(P)
970 RETURN
990 PRINT CHR$(19);
1000 FOR J=1 TO 9:PRINT CHR$(17):NEXT
1010 RETURN
1050 GOSUB 990:PRINT CHR$(17):RETURN
1100 PRINT CHR$(157);" ";
1110 RETURN
1200 PRINT CHR$(147);TAB(16);"W A R I"
1210 PRINT TAB(15);LEFT$(A$,8):PRINT
1220 PRINT TAB(8);"COMPUTER":PRINT
1230 FOR J=0 TO 5:PRINT TAB(4*J+1);12-J;:NEXT
1240 PRINT TAB(30);"CAPTURED"
1245 PRINT A$;TAB(29);LEFT$(A$,9):FOR J=1 TO 2
1250 PRINT B$:PRINT B$:PRINT C$:NEXT:PRINT
1260 FOR J=0 TO 5:PRINT TAB(4*J+1);J+1;:NEXT
1270 PRINT:PRINT:PRINT TAB(11);"YOU"
1280 PRINT:RETURN
2000 PRINT"NO LEGAL MOVES."
2010 PRINT"GAME IS A DRAW."
2020 GOTO 840

READY
```

EASY CHANGES

1. Want a faster moving game against an opponent who isn't quite such a good player? Insert the following two lines:

   ```
   555 GOTO 600
   665 E(A)=V(K):GOTO 690
   ```

   In the standard version of the game, the computer looks at each of its possible moves and each of your possible replies when evaluating which move to make. This change causes the computer to look only at each of its moves, without bothering to look at any of your possible replies. As a result, the computer does not play as well, but it takes only a few seconds to make each move.
2. If you are curious about what the computer thinks are the relative merits of each of its possible moves, you can make this change to find out. Change line 690 so it looks like this:

   ```
   690 PRINT E(A);:NEXT:M=0:FA=-F:FOR J=1 TO 6
   ```

   This will cause the program to display its evaluation number for each of its moves in turn (starting with square seven). It will select the largest number of the six. A negative value means that it will lose stones if that move is made, assuming that you make the best reply you can. A value of negative 50 indicates an illegal move. A positive value greater than one means that a capture can be made by the computer, which will come out ahead after your best reply.

MAIN ROUTINES

120- 155	Initializes variables. Displays board.
160- 180	Asks who goes first. Evaluates answer.
190- 220	Determines computer's move. Displays new board position.
230- 240	Determines if computer's move resulted in a win. Displays a message if so.
250- 300	Gets operator's move. Checks for legality. Displays new board position.
310- 320	Determines if operator's move resulted in a win.
330- 340	Displays message if illegal move attempted.
350- 500	Subroutine to make move M in T array.
360- 390	Copies T array into Y array (inverts if computer is making the move).
400- 430	Makes move in Y array.

440- 450	Checks for captures. Removes stones. Checks previous square.
460- 470	Sees if opponent is left with a legal move.
480- 500	Copies Y array back into T array.
510- 710	Subroutine to determine computer's move.
750- 800	Subroutine to create graphics strings for board display.
810- 880	Displays ending message. Asks about playing again.
900- 970	Subroutine to display stones on board and captured, and "cross-bars" of board squares.
990-1010	Subroutine to move cursor to "YOUR MOVE" position on screen.
1050	Subroutine to move cursor to "MY MOVE" position on screen.
1100-1110	Backspace cursor 1 space and display 1 blank character.
1200-1280	Subroutine to display Wari board (without stones), titles, and square numbers.
2000-2020	Displays message when computer has no legal move.

MAIN VARIABLES

J,K	Subscript variables.
Q,P,F,D	Constant values of 14, 13, 50 and 12, respectively.
T,Y,W	Arrays with temporary copies of the Wari board.
V	Array with evaluation values of operator's six possible replies to computer's move being considered.
E	Array with evaluation values of computer's six possible moves.
B	Array containing Wari board. Thirteenth element has stones captured by operator. Fourteenth has computer's.
WA,WB	Weighting factors for evaluation function.
MN	Move number.
R$	Operator's reply. Also used as switch to indicate whose move it is (C for computer, H for human).
M	Move being made (1-6 for operator, 7-12 for computer). Set negative if illegal.
C	Subscript used in dropping stones around board.
L	Last square in which a stone was dropped.

S	Stones on opponent's side of the board after a move.
A	Subscript used to indicate which of the six possible computer moves is currently being evaluated.
FA	First evaluation factor used in determining favorability of board position after a move (indicates computer's number of occupied squares).
FB	Second evaluation factor (total stones on computer's side of the board).
FC	Third evaluation factor (number of squares with two or less stones).
FD	Fourth evaluation factor (number of stones in most populous square on computer's side).
FE	Total stones on board.
A$,B$,C$	String of graphics characters used to display the Wari board.
D$	String of 32 blanks.

SUGGESTED PROJECTS

1. Modify the program to declare the game a draw if neither player has made a capture in the past thirty moves. Line 300 adds one to a counter of the number of moves made. To make the change, keep track of the move number of the last capture, and compare the difference between it and the current move number with 30.
2. Modify the evaluation function used by the computer strategy to see if you can improve the quality of its play. Lines 600 through 660 examine the position of the board after the move that is being considered. Experiment with the factors and/or the weighting values, or add a new factor of your own.
3. Change the program so it can allow two people to play against each other, instead of just a person against the computer.

Section 4

Graphics Display Programs

INTRODUCTION TO GRAPHICS DISPLAY PROGRAMS

The Commodore 64 is an amazing machine. It has very useful color graphics capabilities in addition to its other capacities. Programs in the other sections of this book take advantage of these graphics to facilitate and "spice up" their various output. Here we explore their use for sheer fun, amusement, and diversion.

Ever look through a kaleidoscope and enjoy the symmetric changing patterns produced? KALEIDO will create such effects to keep you hypnotized.

Two other programs produce ever changing patterns but with much different effects. SPARKLE will fascinate you with a changing shimmering collage. SQUARES uses geometric shapes to obtain its pleasing displays.

WALLOONS demonstrates a totally different aspect of the computer. This program will keep you entertained with an example of computer animation.

KALEIDO

PURPOSE

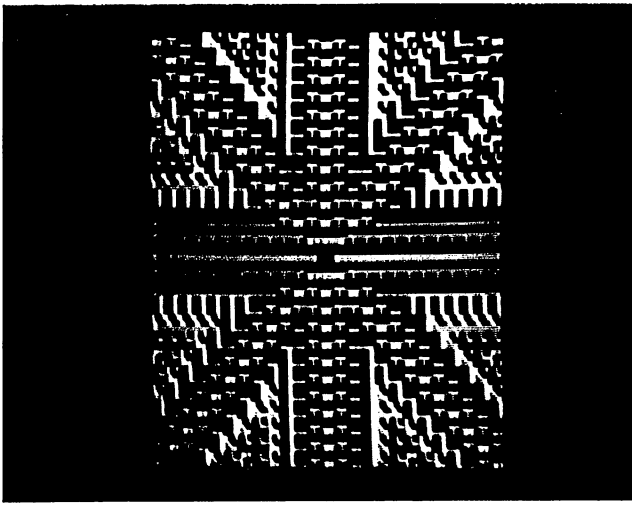
If you have ever played with a kaleidoscope, you were probably fascinated by the endless symmetrical patterns you saw displayed. This program creates a series of kaleidoscope-like designs, with each one overlaying the previous one.

HOW TO USE IT

There is not much to say about how to use this one. Just type RUN, then sit back and watch. Turning down the lights and playing a little music is a good way to add to the effect.

Have a few friends bring their Commodore 64's over (all your friends *do* have Commodore 64's, don't they?), and get them all going with KALEIDO at once. Let us know if you think you have set a new world's record. Please note that we will not be responsible for any hypnotic trances induced this way.

SAMPLE RUN



One of the patterns generated by the KALEIDO program.

PROGRAM LISTING

```
READY.

100 REM: KALEIDO
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 R=RND(-TI):PRINT CHR$(147)
130 A=19:B=12:S=1024:D=-1:C=55296
135 M=63:POKE 53280,1:POKE 53281,0
140 DIM R(6)
150 FOR J=0 TO 6
160 T=65
165 IF RND(1)>.5 THEN T=T+128
170 R(J)=INT(M*RND(1))+T:NEXT
180 D=1:K=1:L=12:IF D<0 THEN K=12:L=1
200 FOR J=K TO L STEP D
210 X=A+J:Y=B:GOSUB 900
220 X=A-J:GOSUB 900
230 X=A:Y=B+J:GOSUB 900
240 Y=B-J:GOSUB 900
```

```
250 X=A+J:Y=B+J:GOSUB 900
260 X=A-J:Y=B-J:GOSUB 900
270 Y=B+J:GOSUB 900
280 X=A+J:Y=B-J:GOSUB 900
790 NEXT
795 FOR J=1 TO 2000:NEXT J
800 GOTO 150
900 C1=INT(RND(1)*14)+1
905 POKE C+40*Y+X,C1
910 POKE S+40*Y+X,R(0)
915 IF J=1 THEN RETURN
920 W=INT(J*.5):T=J-W-1
930 FOR N=1 TO W
940 IF X=A THEN Y2=Y:X2=X+N:GOSUB 2000
945 IF X=A THEN X2=X-N:GOSUB 2000:NEXT:RETURN
950 IF Y=B THEN X2=X:Y2=Y+N:GOSUB 2000
955 IF Y=B THEN Y2=Y-N:GOSUB 2000:NEXT:RETURN
970 Y2=Y:IF X<A THEN X2=X+N:GOSUB 2000:GOTO 990
980 X2=X-N:GOSUB 2000
990 X2=X:IF Y<B THEN Y2=Y+N:GOSUB 2000:GOTO 1010
1000 Y2=Y-N:GOSUB 2000
1010 NEXT
1020 RETURN
2000 POKE S+40*Y2+X2,R(N)
2005 POKE C+40*Y2+X2,C1
2010 RETURN

READY.
```

EASY CHANGES

1. Change the first part of line 180 to say D = – D instead of D = 1. This will cause alternating inward and outward drawing of the designs rather than always outward from the center.
2. In line 795, change the constant of 2000 to 10000. This will cause a delay of about fifteen seconds between the drawing of successive design instead of three seconds. Or, remove line 795 to eliminate the delay entirely.
3. Modify the range of graphics characters from which the ones in the design are randomly selected. This is done by modifying the values of M and T in lines 135 and 160. For example, try

	M = 5 and T = 76
or	M = 30 and T = 76
or	M = 12 and T = 116

Experiment with other values. Be sure that M and T are both positive integers, that T is at least 65, and that the sum of M and T is no greater than 128.

4. Eliminate reverse graphics characters by deleting line 165. Or, lower the frequency of use of the reverse graphics characters by increasing the value of .5 in line 165 to, say, .8 or .9. Similarly, you can increase the frequency of use of reverse graphics characters by lowering the .5 to .1 or .2.
5. You can change the designs into ones with only four point symmetry by doing the following:
   1. In lines 140 and 150, change the 6 to a 12.
   2. In line 930, change the W to a J.
   3. Insert this line

```
245 GOTO 790
```

MAIN ROUTINES

120- 140	Housekeeping. Initializes variables, RND.
150- 170	Picks seven random graphics characters.
180	Selects D, K, and L to draw patterns inward or outward.
200- 790	Mainline routine. Determines axes of pattern.
795	Delays about three seconds after drawing design.
900-1020	POKEs graphics character into axes of design, then determines coordinates of points between axes. Generates random color.
2000-2010	POKEs characters and colors between axes (subroutine).

MAIN VARIABLES

A,B	Coordinates of center of design. Upper left corner is considered to be (0,0).
S	Starting address of CRT memory map area.
D	Direction in which design is drawn (1 = outward, − 1 = inward).
R	Array for the seven random graphics characters.
J,N	Subscript variables.
T	Numeric representation of lowest graphics character to be used.
K,L	Inner and outer bounds of design (distance from center).

X,Y	Coordinates of character being POKEd on horizontal, vertical, or diagonal axes.
X2,Y2	Coordinates of character being POKEd between horizontal, vertical, or diagonal axes.
M	Number of possible graphics characters to choose from, not counting reverses.
C	Starting address of CRT color map.
C1	Random color of graphics character.

SPARKLE

PURPOSE

This graphics display program provides a continuous series of hypnotic patterns, some of which seem to sparkle at you while they are created. Two types of patterns are used. The first is a set of colored concentric diamond shapes in the center of the screen. Although the pattern is somewhat regular, the sequence in which it is created is random, which results in the "sparkle" effect.

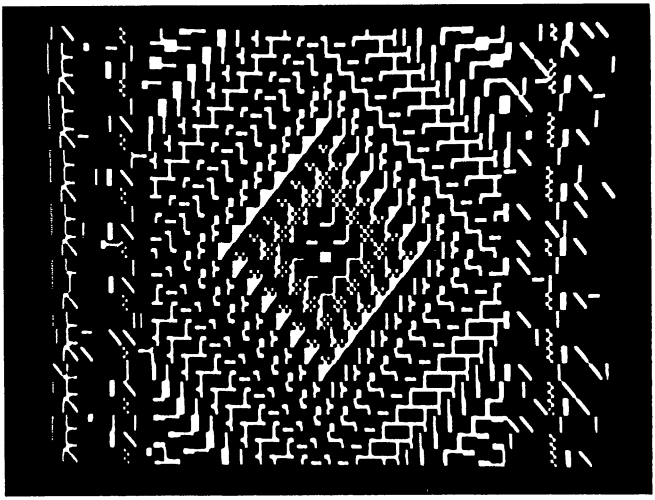
The second type of pattern starts about two seconds after the first has finished. It is a series of "sweeps" across the screen—left to right and top to bottom. Each sweep uses a random graphics color that is spaced equally across the screen. The spacing distance is chosen at random for each sweep. Also, the number of sweeps to be made is chosen at random each time in the range from 11 to 30.

After the second type of pattern is complete, the program goes back to the first type, which begins to overlay the second type.

HOW TO USE IT

Confused by what you just read? Never mind. You have to see it to appreciate it. Just enter the program into your Commodore 64, then sit back and watch the results of your labor.

SAMPLE RUN



One of the patterns generated by the SPARKLE program.

PROGRAM LISTING

```
READY.

100 REM: SPARKLE
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 R=RND(-TI):PRINT CHR$(147)
125 POKE 53280,1:POKE 53281,0
130 DIM A(12),B(12):A=19:B=12:S=1024:C1=55296
140 T=INT(37*RND(1))+66
150 FOR J=0 TO 12:A(J)=J:B(J)=J:NEXT
160 FOR J=0 TO 12:R=INT(13*RND(1))
170 W=A(J):A(J)=A(R):A(R)=W:NEXT
180 FOR J=0 TO 12:R=INT(13*RND(1))
190 W=B(J):B(J)=B(R):B(R)=W:NEXT
200 FOR J=0 TO 12:FOR K=0 TO 12
210 R=A(J):W=B(K):C=R+W+T
240 X=A+R:Y=B+W:GOSUB 900
```

```
250 Y=B-W:GOSUB 900
260 X=A-R:GOSUB 900
270 Y=B+W:GOSUB 900
280 X=A+W:Y=B+R:GOSUB 900
290 Y=B-R:GOSUB 900
300 X=A-W:GOSUB 900
310 Y=B+R:GOSUB 900
320 NEXT:NEXT
350 FOR J=1 TO 2000:NEXT
400 N=65:M=63
410 FOR J=1 TO INT(21*RND(1))+10
420 R=INT(22*RND(1))+1:W=INT(M*RND(1))
430 T=S:IF RND(1)>.8 THEN T=T+1
432 C2 = INT(RND(1)*14)+1
434 POKE C1,C2
440 POKE S,N+W
450 FOR K=T TO T+999 STEP R
452 C2 = INT(RND(1)*14)+1
455 POKE C1+(K-T),C2
460 POKE K,N+W:NEXT
470 NEXT
480 GOTO 140
900 C2 = INT(RND(1)*14)+1
920 POKE C1+40*Y+X,C2
930 POKE S+40*Y+X,C
940 RETURN

READY.
```

EASY CHANGES

1. Make the second type of pattern appear first by inserting this line:

   135 GOTO 400

   Or, eliminate the first type of pattern by inserting:

   145 GOTO 400

   Or, eliminate the second type of pattern by inserting:

   360 GOTO 140

2. Increase the delay after the first type of pattern by increasing the 2000 in line 350 to, say, 5000. Remove line 350 to eliminate the delay.
3. Increase the number of sweeps across the screen of the second type of pattern by changing the 10 at the right end of line 410 into a 30 or a 50, for example. Decrease the number

of sweeps by changing the 10 to a 1, and also changing the 21 in line 410 to 5 or 10.

4. Watch the effect on the second type of pattern if you change the 22 in line 420 into various integer values between 2 and 100.
5. Change the value of N and M in line 400 to alter the graphics characters used in the second type of pattern. For example, try

   M=5 and N=76

   Be sure N is at least 65 and the sum of N and M is no more than 128.

MAIN ROUTINES

120-130	Initializes variables. Clears screen.
140-320	Displays square pattern in center of screen.
150-190	Shuffles the numbers 0 through 15 in the A and B arrays.
200-320	POKEs graphics characters to the screen.
350	Delays for about 2 seconds.
400-480	Overlays the entire screen with a random graphics character spaced at a fixed interval chosen at random.
900-940	Subroutine to POKE graphics character C to location (X,Y). Upper left corner of screen is location (0,0).

MAIN VARIABLES

R	Random integer. Also, work variable.
A,B	Arrays in which shuffled integers from 0 to 12 are stored for use in making first type of pattern.
A,B	Coordinates of center of screen (19 across, 12 down).
S	Starting address of CRT memory area.
T	Integer from 66 to 102, used in creating random graphics characters. Also, work variable.
J,K	Work and loop variables.
W	Work variable.
X,Y	Coordinates of a position on the screen for a graphics character.

C	Graphics character to be POKEd to screen at X,Y.
N	Lowest graphics character to be used in second type of pattern.
M	Multiplier used in getting a random integer to add to N.
C1	Starting location of color map.
C2	Color for current graphics character.

SUGGESTED PROJECTS

Make the second type of pattern alternate between "falling from the top" (as it does now) and rising from the bottom of the screen.

SQUARES

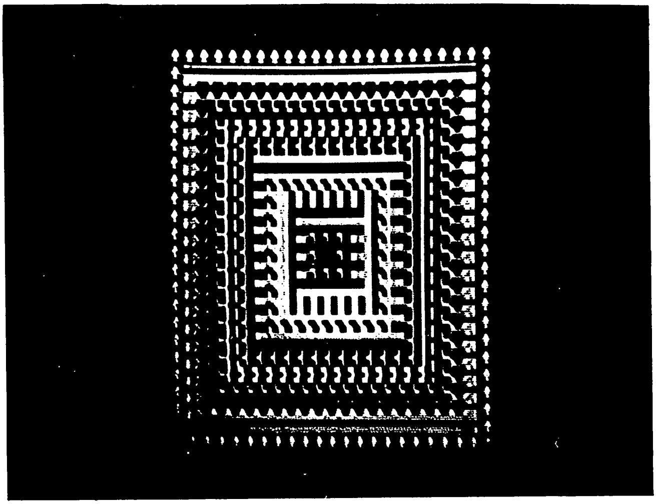
PURPOSE

This is another graphics-display program. It draws a series of concentric squares with the graphics color used for each one chosen at random. After a full set of concentric squares is drawn, the next set starts again at the center and overlays the previous one. They are actually rectangles, not squares, but let's not be nit-pickers.

HOW TO USE IT

As with most of the other graphics display programs, you just sit back and enjoy watching this one once you get it started.

SAMPLE RUN



One of the patterns generated by the SQUARES program.

PROGRAM LISTING

```
READY.

100 REM: SQUARES
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 J=RND(-TI):PRINT CHR$(147)
121 POKE 53280,1:POKE 53281,0
122 C1(0)=5:C1(1)=28:C1(2)=30:C1(3)=156
124 C1(4)=158:C1(5)=159
130 GOSUB 600:N=1:R=INT(50*RND(1))
140 C=INT(95*RND(1))+160
150 PRINT CHR$(146);
160 IF RND(1)>.5 THEN PRINT CHR$(18);
170 X=C:Y=0:K=N:GOSUB 700
180 N=N+1
190 Y=145:Z=157:K=N:GOSUB 800
200 Y=157:GOSUB 800
210 N=N+1
```

```
220 Y=17:K=N:GOSUB 800
225 C2=INT(RND(1)*6):PRINT CHR$(C1(C2));
230 IF N<22 THEN 140
240 GOTO 130
600 PRINT CHR$(19)
610 FOR K = 1 TO 20:PRINT CHR$(29);:NEXT
620 FOR K = 1 TO 12:PRINT CHR$(17);:NEXT
630 RETURN
700 FOR J=1 TO K:GOSUB 900
710 PRINT CHR$(X);CHR$(Y);:NEXT:RETURN
800 FOR J=1 TO K:GOSUB 900
810 PRINT CHR$(X);CHR$(Y);CHR$(Z);
820 NEXT:RETURN
900 FOR W=1 TO R:NEXT:RETURN

READY.
```

EASY CHANGES

1. To eliminate the variable speed feature of the program, insert this line:

   135 R=25

   This will cause the speed to be fixed at about the level of the average speed when the variable speed feature is active. Instead of 25, try 1 for high speed, or 50 or 100 for low speed.
2. To eliminate the use of reverse graphics characters in the display, remove line 160.
3. To make the patterns less regular, try inserting these lines:

   802 D=0:IF INT(J/2)=J/2 THEN D=1
   804 X=X+D
   815 X=X−D

MAIN ROUTINES

120-130	Initializes variables. Clears screen. Determines speed for this set of squares.
140-160	Picks random graphics character.
170-220	Mainline routine. Draws a square using cursor control characters.
230-240	Determines if set of squares is done. If so, starts over. If not, draws next square.
600-610	Subroutine to move cursor to center of screen.

700-710	Subroutine to do K repetitions of printing CHR$ of X and Y.
800-820	Subroutine to do K repetitions of printing CHR$ of X, Y, and Z.
900	Delay subroutine.

MAIN VARIABLES

J,K,W	Work and loop variables.
N	Length of the side currently being drawn.
R	Random number from 0 to 49 for time delay.
C	ASCII values of graphics character.
X,Y,Z	ASCII values of characters being printed by subroutines. Can be cursor control characters or graphics characters.
C1	Array of CRT ASCII colors to use for each square.
C2	ASCII color code for current square.

WALLOONS

PURPOSE

The Commodore 64 is quite a versatile machine. This program takes advantage of its powerful graphics capability to produce computer animation. That's right, animation! WALLOONS will entertain you with a presentation from the Color Circus.

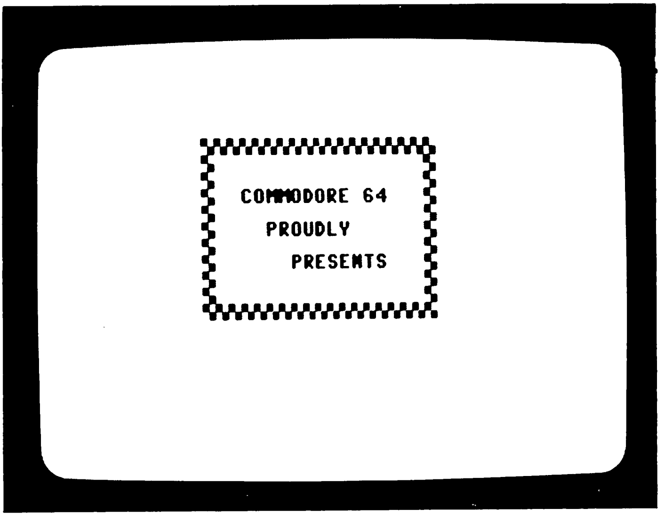
The Color Circus searches the world over to bring you the best in circus acts and other performing artists. Today, direct from their performance before the uncrowned heads of Europe, the Circus brings you the Flying Walloons.

HOW TO USE IT

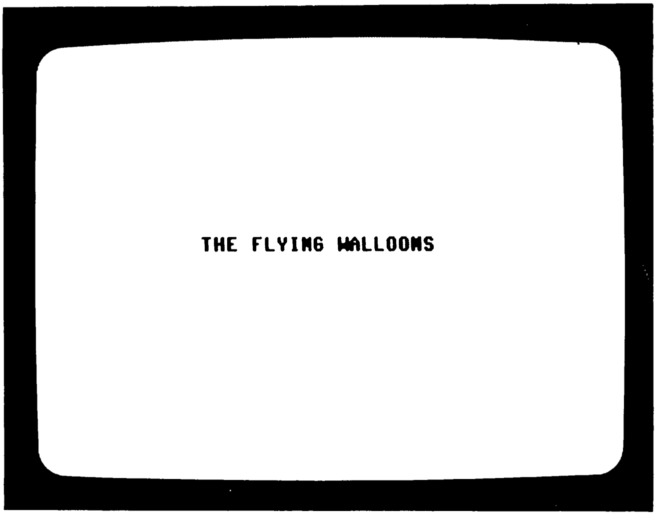
Just sit back, relax, and get ready to enjoy the show. Type RUN and the Flying Walloons will be ready to perform. You have a front row center seat and the show is about to begin.

Applause might be appropriate if you enjoy their performance. Please note that the Walloons have been working on a big new finish to their act which they haven't yet quite perfected.

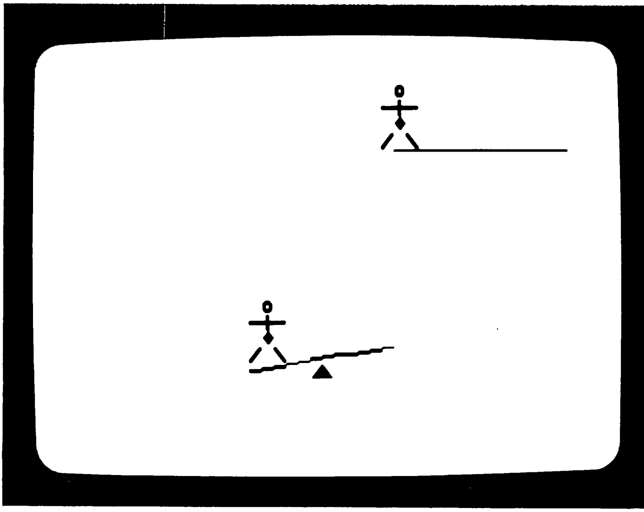
SAMPLE RUN



The billboard announces a new presentation of the (in)famous Commodore 64.



The Flying Walloons are to perform!



The Walloons attempt a dangerous trick from their repertoire.

PROGRAM LISTING

```
READY.

100 REM: WALLOONS
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 NJ=3
160 POKE 53280,2:POKE 53281,9
190 PRINT CHR$(158);CHR$(142)
200 GOSUB 1900:FOR J=1 TO 7:PRINT:NEXT
210 A=1234:FOR J=0 TO 18:POKE A+J,127
220 NEXT:FOR J=0 TO 10:POKE A+18+40*J,127
230 NEXT:FOR J=0 TO 10:POKE A+40*J,127
240 NEXT:FOR J=0 TO 18:POKE A+400+J,127
250 FOR J=0 TO 18:POKE A+400+J,127:NEXT
260 GOSUB 1920
270 PRINT TAB(13);"COMMODORE 64":PRINT
280 PRINT TAB(15);"PROUDLY":PRINT
290 PRINT TAB(17);"PRESENTS"
400 GOSUB 1920:GOSUB 1920:PRINT CHR$(19)
410 FOR J=1 TO 9:PRINT CHR$(17);:NEXT
420 FOR J=1 TO 13:PRINT CHR$(29);:NEXT
```

```
430 FOR J=1 TO 13:PRINT"-";
435 FOR K=1 TO 300:NEXT:NEXT
440 GOSUB 1920:GOSUB 1920
450 GOSUB 1900:FOR J=1 TO 10:PRINT CHR$(17);
460 NEXT:PRINT TAB(10);"THE FLYING WALLOONS"
470 GOSUB 1920:GOSUB 1920
500 A=1639:GOSUB 1900:GOSUB 1700
510 GOSUB 1920:GOSUB 1920
520 FOR A=1626 TO 1638:GOSUB 1800
530 GOSUB 1850:NEXT:GOSUB 1800
600 GOSUB 1920
610 A=1132:R=0
620 FOR Q=A+158 TO A+171:POKE Q,99:NEXT
630 FOR L=1 TO 2000:NEXT
640 A=1142:GOSUB 1800:GOSUB 1920
650 FOR A=1142 TO 1132 STEP -1
660 FOR L=1 TO 100:NEXT
670 GOSUB 1800:GOSUB 1850:NEXT:GOSUB 1800
680 GOSUB 1920:GOSUB 1850:A=A-1:GOSUB 1800
690 GOSUB 1920
700 GOSUB 1920:GOSUB 1850:A=A+1:GOSUB 1800
710 GOSUB 1850:A=A+1:GOSUB 1800:GOSUB 1920
720 POKE A,81:GOSUB 1920:GOSUB 1920:
730 POKE A,87:GOSUB 1920:GOSUB 1920
740 FOR A=1132 TO 1128 STEP-1
750 GOSUB 1800:GOSUB 1850
760 FOR L=1 TO 100:NEXT:NEXT:GOTO 900
800 A=1639:GOSUB 1900:GOSUB 1800
810 GOSUB 1700
820 FOR A=1606 TO 1128 STEP -40
830 GOSUB 1800:GOSUB 1850:NEXT
840 R=R+1:IF R>=NJ THEN 1000
900 FOR A=1128 TO 1608 STEP 40
910 GOSUB 1800:GOSUB 1850:NEXT
920 A=1648:GOSUB 1900:GOSUB 1800
930 GOSUB 1600
940 FOR A= 1599 TO 1119 STEP -40
950 GOSUB 1800:GOSUB 1850:NEXT
960 FOR A= 1119 TO 1599 STEP 40
970 GOSUB 1800:GOSUB 1850:NEXT
980 GOTO 800
1000 A=A-39:GOSUB 1800:GOSUB 1850
1010 A=A+41:GOSUB 1800:GOSUB 1850
1020 FOR A= 1090 TO 1610 STEP 40
1030 GOSUB 1800:GOSUB 1850:NEXT
1040 A= 1614:POKE A,87:POKE A-1,91
1050 POKE A-41,93:POKE A+39,93:
1060 POKE A-2,90:POKE A-43,77
1070 POKE A+37,78
1080 GOSUB 1920:GOSUB 1920:GOSUB 1900
```

```
1090 FOR J=1 TO 5:PRINT CHR$(17):NEXT
1100 PRINT TAB(18);"FINIS":GOSUB 1920
1110 GOSUB 1920:GOSUB 1920:END
1600 POKE A+110,99:POKE A+111,69
1610 POKE A+112,68:POKE A+113,67
1620 POKE A+114,64:POKE A+115,70
1630 POKE A+116,82:POKE A+117,100
1640 POKE A+158,99:POKE A+159,69
1650 POKE A+160,68:POKE A+161,67
1660 POKE A+155,233:POKE A+156,223
1670 RETURN
1700 POKE A+123,100:POKE A+124,82
1710 POKE A+125,70:POKE A+126,64
1720 POKE A+127,67:POKE A+128,68
1730 POKE A+129,69:POKE A+130,99
1740 POKE A+159,67:POKE A+160,68
1750 POKE A+161,69:POKE A+162,99
1760 POKE A+164,233:POKE A+165,223:RETURN
1800 POKE A,87:POKE A+39,64:POKE A+40,91
1810 POKE A+41,64:POKE A+80,90
1820 POKE A+119,78:POKE A+121,77:RETURN
1850 POKE A,32:POKE A+39,32:POKE A+40,32
1860 POKE A+41,32:POKE A+80,32
1870 POKE A+119,32:POKE A+121,32:RETURN
1900 POKE 53281,1:PRINT CHR$(147)
1901 POKE 53281,9:RETURN
1920 FOR L=1 TO 1000:NEXT:RETURN

READY.
```

EASY CHANGES

1. If you wish to have the Walloons perform more (or less) jumps during their performance, change the value of NJ in line 150 accordingly. To get five jumps, use

   150 NJ = 5

2. Timing delays are used often in the program. To change the length of the delay, alter the 1000 in line 1920 to a different value. Values larger than 1000 will lengthen the delays, while values smaller than 1000 will shorten the delays.

3. You might want to personalize the title placard and make yourself the presenter of the Walloons. This can be done by altering the string literal, "COMMODORE 64" in line 270 to something else. However, you cannot use a string with a length of much more than 14 characters or it will print

beyond the end of the placard. To say, for example, that Simon Fenster presents the Walloons, change line 270 to:

```
270 PRINT TAB(13);"SIMON FENSTER":PRINT
```

MAIN ROUTINES

150	Sets NJ.
200- 290	Displays title placard.
400- 470	Removes "proudly," displays rest of title.
500- 530	Moves first Walloon into view.
600- 760	Second Walloon enters from the high platform.
800- 980	Flying Walloons perform.
1000-1110	Concludes Walloon's performance.
1600-1670	Subroutine to draw lever with right side down.
1700-1760	Subroutine to draw lever with right side up.
1800-1820	Subroutine to draw Walloon with head at Poke location A.
1850-1870	Subroutine to erase Walloon with head at A.
1900-1901	Subroutine to clear screen.
1920	Time delaying subroutine.

MAIN VARIABLES

NJ	Number of jumps to make.
A	Reference Poke location.
R	Jump Counter.
J,K,L,Q	Loop indices.

SUGGESTED PROJECTS

1. There are many possibilities for "spicing up" the Walloons' act with extra tricks or improved ones. Perhaps you would like to change their finish to something less crude.
2. If you add some alternate tricks or endings as suggested in the previous project, try randomizing if and when they will be done. Thus, the Walloon's performance will be different each time the program is run. At least their ending may be variable.
3. Scour the world yourself for other acts to include in the Commodore 64 Circus. Maybe someday we will have a complete software library of performing artists.

Section 5

Mathematics Programs

INTRODUCTION TO MATHEMATICS PROGRAMS

Since their invention, computers have been used to solve mathematical problems. Their great speed and reliability render solvable many otherwise difficult (or impossible) calculations. Several different numerical techniques lend themselves naturally to computer solution. The following programs explore some of them. They will be of interest mainly to engineers, students, mathematicians, statisticians, and others who encounter such problems in their work.

GRAPH takes advantage of the Commodore 64's graphic powers to draw the graph of a function $Y = f(X)$. The function is supplied by you. INTEGRATE calculates the integral, or "area under the curve," for any such function.

Experimental scientific work frequently results in data at discrete values of X and Y. CURVE finds a polynomial algebraic expression to express this data with a formula.

Theoretical scientists (and algebra students) often must find the solution to a set of simultaneous linear algebraic equations. SIMEQN does the trick.

Much modern engineering work requires the solution of differential equations. DIFFEQN will solve any first-order ordinary differential equation that you provide.

STATS will take a list of data and derive standard statistical information describing it. In addition, it will sort the data list into ranking numerical order.

CURVE

PURPOSE AND DISCUSSION

CURVE fits a polynomial function to a set of data. The data must be in the form of pairs of X-Y points. This type of data occurs frequently as the result of some experiment, or perhaps from sampling tabular data in a reference book.

There are many reasons why you might want an analytic formula to express the functional relationship inherent in the data. Often you will have experimental errors in the Y values. A good formula expression tends to smooth out these fluctuations. Perhaps you want to know the value of Y at some X not obtained exactly in the experiment. This may be a point between known X values (interpolation) or one outside the experimental range (extrapolation). If you wish to use the data in a computer program, a good formula is a convenient and efficient way to do it.

This program fits a curve of the form

$$Y = C_0 + C_1X^1 + C_2X^2 + \ldots + C_DX^D$$

to your data. You may select D, the degree (or power) of the highest term, to be as large as 9. The constant coefficients, C_0-C_D, are the main output of the program. Also calculated is the goodness of fit, a guide to the accuracy of the fit. You may fit different degree polynomials to the same data and also ask to have Y calculated for specific values of X.

The numerical technique involved in the computation is known as least squares curve fitting. It minimizes the sum of the squares of the errors. The least squares method reduces the

problem to a set of simultaneous algebraic equations. Thus these equations could be solved by the algorithm used in SIMEQN. In fact, once the proper equations are set up, CURVE uses the identical subroutine found in SIMEQN to solve the equations. For more information, the bibliography contains references to descriptions of the numerical technique.

HOW TO USE IT

The first thing you must do, of course, is enter the data into the program. This consists of typing in the pairs of numbers. Each pair represents an X value and its corresponding Y value. The two numbers (of each pair) are separated by a comma. A question mark will prompt you for each data pair. After you have entered them all, type

999,999

to signal the end of the data. When you do this, the program will respond by indicating how many data pairs have been entered. A maximum of 100 data pairs is allowed.

Next, you must input the degree of the polynomial to be fitted. This can be any non-negative integer subject to certain constraints. The maximum allowed is 9. Unless your data is well behaved (X and Y values close to 1), the program will often not produce accurate results if D is greater than 5 or so. This is because the sums of powers of X and Y are calculated up to the powers of 2*D. These various sums are several orders of magnitude different than each other. Errors result because of the numerous truncation and round-off operations involved in doing arithmetic with them. Also, D must be less than the number of data pairs. You will get an error message if you input an illegal value of D.

A few notes regarding the selection of D may be of interest. If $D = 0$, the program will output the mean value of Y as the coefficient C_0. If $D = 1$, the program will be calculating the best straight line through the data. This special case is known as "linear regression." If D is one less than the number of data pairs, the program will find an exact fit to the data (barring round-off and other numerical errors). This is a solution which passes exactly through each data point.

Once you have entered the desired degree, the program will begin calculating the results. There will be a pause while this calculation is performed. The time involved depends on the number of data pairs and the degree selected. For twenty-five data pairs and a third degree fit, the pause will be about half a minute. Fifty data pairs and a fifth degree fit will take about a minute.

The results are displayed in a table. It gives the values of the coefficients for each power of X from 0 to D. That is, the values of C_0-C_D are output. Also shown is the percent goodness of fit. This is a measure of how accurately the program was able to fit the given case. A value of 100 percent means perfect fit, lesser values indicate correspondingly poorer fits. It is hard to say what value denotes *satisfactory* fit since much depends on the accuracy of data and the purpose at hand. But as a rule of thumb, anything in the high nineties is quite good. For those interested, the formula to calculate the percent goodness of fit is

$$\text{P.G.F.} = 100^* \sqrt{1 - \frac{\sum_i (Y_i - \hat{Y}_i)^2}{\sum_i (Y_i - \bar{Y})^2}}$$

where Y_i are the actual Y data values, $\hat{Y}_i$ are the calculated Y values (through the polynomial expression), and $\bar{Y}$ is the mean value of Y.

Next, you are presented with three options for continuing the run. These are 1) determining specific points, 2) fitting another degree, 3) ending the program. Simply type 1, 2, or 3 to make your selection. A description of each choice now follows.

Option 1 allows you to see the value of Y that the current fit will produce for a given value of X. In this mode you are continually prompted to supply any value of X. The program then shows what the polynomial expression produces as the value for Y. Input 999 for an X value to leave this mode.

Option 2 allows you to fit another degree polynomial to the same data. Frequently, you will want to try successively higher values of D to improve the goodness of fit. Unless round-off errors occur, this will cause the percent goodness of fit to increase.

Option 3 simply terminates the program and with that we will terminate this explanation of how to use CURVE.

SAMPLE PROBLEM AND RUN

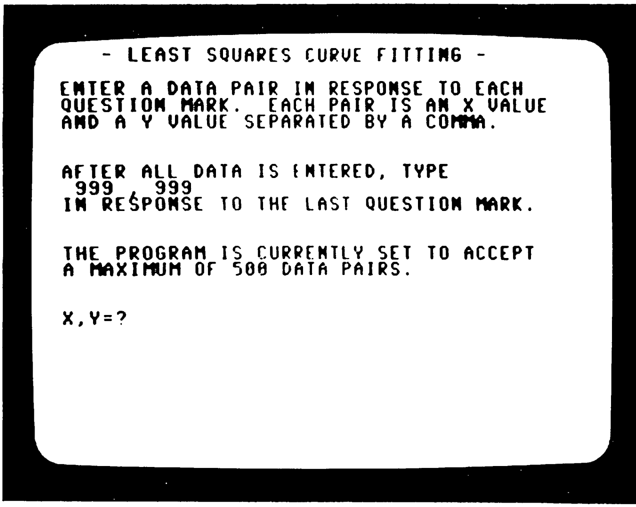
Problem: An art investor is considering the purchase of Primo's masterpiece, "Frosted Fantasy." Since 1940, the painting has been for sale at auction seven times. Here is the painting's sales record from these auctions.

Year	Price
1940	$ 8000.
1948	$13000.
1951	$16000.
1956	$20000.
1962	$28000.
1968	$39000.
1975	$53000.

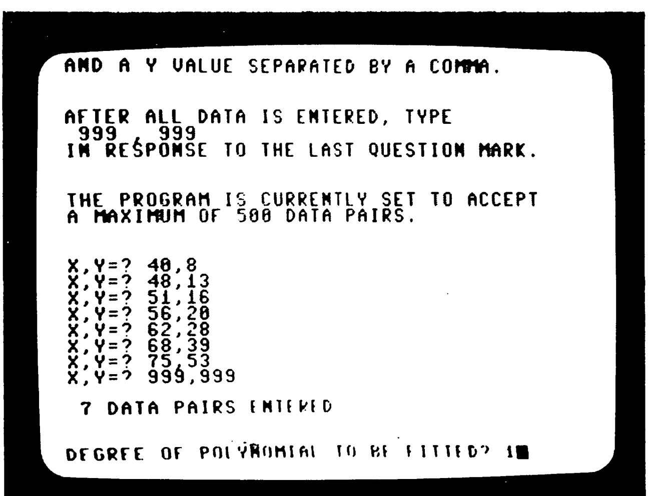
The painting is going to be sold at auction in 1983. What price should the investor expect to have to pay to purchase the painting? If he resold it in 1986, how much profit should he expect to make?

Solution: The investor will try to get a polynomial function that expresses the value of the painting as a function of the year. This is suitable for CURVE. The year will be represented by the variable X, and the price is shown by the variable Y. To keep the magnitude of the numbers small, the years will be expressed as elapsed years since 1900, and the price will be in units of $1000. (thus a year of 40 represents 1940, a price of 8 represents $8000.)

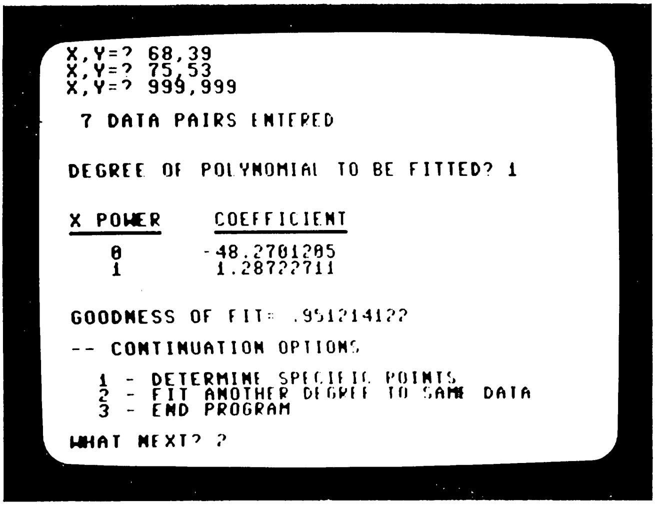
SAMPLE RUN



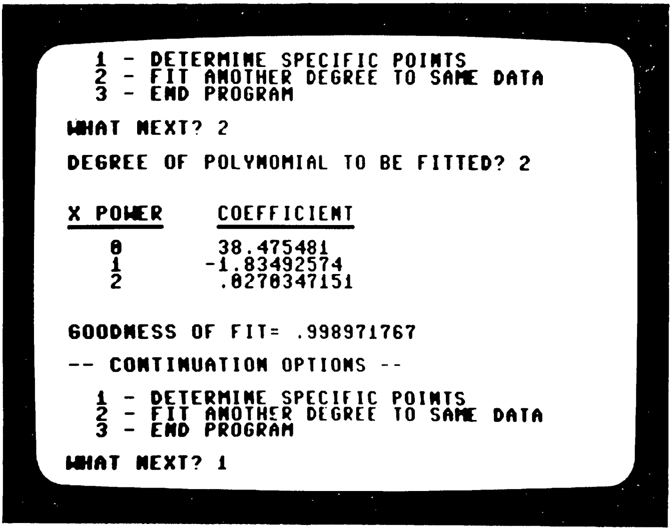
The program displays an introduction and waits for the operator to begin entering the data.



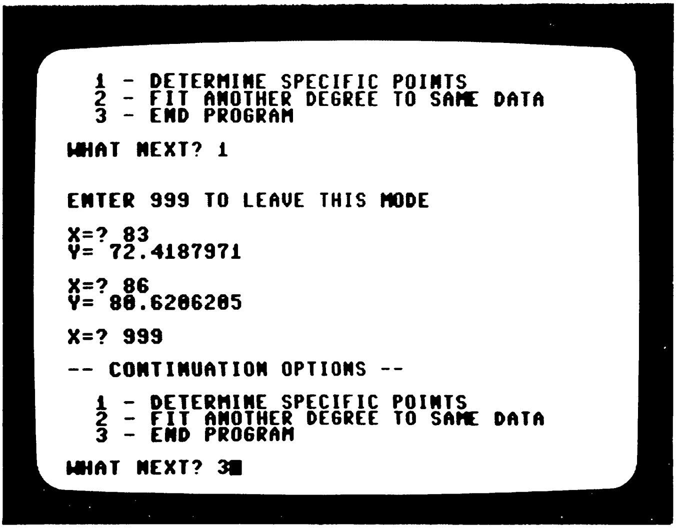
After the operator enters the data, he tries a first degree fit.



The operator obtains a goodness fit of 95.1%. He wants to do better, so he tries for another fit by selecting option 2.



This second degree has a very high goodness of fit. The operator then wants to extrapolate his data to the years 1983 and 1986. He selects option 1.



The operator found that he should expect to pay about $72,400 to buy the painting in 1983. Around an $8200 profit could be expected upon resale in 1986. The operator ends the program.

Of course, the investor did not make his decision solely on the basis of this program. He used it only as one guide to his decision. There is never any guarantee that financial data will perform in the future as it has done in the past. Though CURVE is probably as good a way as any, extrapolation of data can never be a totally reliable process.

PROGRAM LISTING

```
READY.

100 REM: CURVE
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
```

```
150 MX=500
160 EF=999
170 MD=9
200 DIM X(MX),Y(MX)
210 Q=MD+1:DIM A(Q,Q),R(Q),V(Q)
220 Q=MD*2:DIM P(Q)
300 PRINT CHR$(147);"    - LEAST SQUARES ";
305 PRINT "CURVE FITTING -":PRINT
310 PRINT"ENTER A DATA PAIR IN RESPONSE TO EACH"
320 PRINT"QUESTION MARK.  EACH PAIR IS AN X VALUE"
330 PRINT"AND A Y VALUE SEPARATED BY A COMMA."
335 PRINT
340 PRINT:PRINT"AFTER ALL DATA IS ENTERED, TYPE"
350 PRINT EF;",";EF
360 PRINT"IN RESPONSE TO THE LAST QUESTION MARK."
365 PRINT:PRINT
370 PRINT"THE PROGRAM IS CURRENTLY SET TO ACCEPT"
380 PRINT"A MAXIMUM OF";MX;"DATA PAIRS."
400 PRINT:PRINT:J=0
410 J=J+1:INPUT"X,Y=";X(J),Y(J)
420 IF X(J)=EF AND Y(J)=EF THEN J=J-1:GOTO 450
430 IF J=MX THEN PRINT:PRINT"NO MORE DATA ALLOWED"
435 IF J=MX THEN GOTO 450
440 GOTO 410
450 NP=J:PRINT
460 IF NP=0 THEN PRINT"** FATAL ERROR **";
465 IF NP=0 THEN PRINT" -- NO DATA ENTERED":STOP
470 PRINT NP;"DATA PAIRS ENTERED":PRINT
500 PRINT:PRINT"DEGREE OF POLYNOMIAL";
505 INPUT " TO BE FITTED";D:PRINT
510 IF D<0 THEN GOSUB 515:GOTO 500
512 GOTO 520
515 PRINT"** ERROR! ** -- DEGREE MUST BE >= 0"
517 RETURN
520 D=INT(D):IF D<NP THEN 540
530 PRINT"** ERROR! ** -- NOT ENOUGH DATA"
535 GOTO 500
540 D2=2*D:IF D>MD THEN PRINT"** ERROR! ** ";
545 IF D>MD THEN PRINT"-- DEGREE TOO HIGH":GOTO
    500
550 N=D+1
600 FOR J=1 TO D2:P(J)=0:FOR K=1 TO NP
610 P(J)=P(J)+X(K)↑J:NEXT:NEXT:P(0)=NP
620 R(1)=0:FOR J=1 TO NP:R(1)=R(1)+Y(J)
630 NEXT:IF N=1 THEN 660
640 FOR J=2 TO N:R(J)=0:FOR K=1 TO NP
650 R(J)=R(J)+Y(K)*X(K)↑(J-1):NEXT:NEXT
660 FOR J=1 TO N:FOR K=1 TO N:A(J,K)=P(J+K-2)
665 NEXT:NEXT
670 GOSUB 2000
700 PRINT:PRINT"X POWER     COEFFICIENT"
710 FOR J=1 TO 7:PRINT CHR$(197);:NEXT
```

```
715 PRINT TAB(11);
720 FOR J=1 TO 11:PRINT CHR$(197);:NEXT:PRINT
730 FOR J=1 TO N:PRINT"  ";J-1,V(J):NEXT
735 PRINT:PRINT
740 Q=0:FOR J=1 TO NP:Q=Q+Y(J):NEXT
745 M=Q/NP:T=0:G=0:FOR J=1 TO NP
750 Q=0:FOR K=1 TO N:Q=Q+V(K)*X(J)↑(K-1):NEXT
755 T=T+(Y(J)-Q)↑2
760 G=G+(Y(J)-M)↑2:NEXT:IF G=0 THEN T=1:GOTO 780
770 T=1-T/G
780 PRINT"GOODNESS OF FIT=";T
800 PRINT:PRINT"-- CONTINUATION OPTIONS --":PRINT
810 PRINT"  1 - DETERMINE SPECIFIC POINTS"
820 PRINT"  2 - FIT ANOTHER DEGREE TO SAME DATA"
830 PRINT"  3 - END PROGRAM":PRINT
840 INPUT"WHAT NEXT";Q:Q=INT(Q):IF Q=3 THEN END
850 IF Q=2 THEN 500
860 IF Q<>1 THEN 800
900 PRINT:PRINT
905 PRINT"ENTER";EF;"TO LEAVE THIS MODE"
910 PRINT:INPUT"X=";XV:IF XV=EF THEN 800
920 YV=0:FOR K=1 TO N
930 YV=YV+V(K)*XV↑(K-1):NEXT:PRINT"Y=";YV
940 GOTO 910
2000 IF N=1 THEN V(1)=R(1)/A(1,1):RETURN
2010 FOR K=1 TO N-1
2020 I=K+1
2030 L=K
2040 IF ABS(A(I,K))>ABS(A(L,K)) THEN L=I
2050 IF I<N THEN I=I+1:GOTO 2040
2060 IF L=K THEN 2100
2070 FOR J=K TO N:Q=A(K,J):A(K,J)=A(L,J)
2080 A(L,J)=Q:NEXT
2090 Q=R(K):R(K)=R(L):R(L)=Q
2100 I=K+1
2110 Q=A(I,K)/A(K,K):A(I,K)=0
2120 FOR J=K+1 TO N:A(I,J)=A(I,J)-Q*A(K,J):NEXT
2130 R(I)=R(I)-Q*R(K):IF I<N THEN I=I+1:GOTO 2110
2140 NEXT
2150 V(N)=R(N)/A(N,N):FOR I=N-1 TO 1 STEP -1
2160 Q=0:FOR J=I+1 TO N:Q=Q+A(I,J)*V(J)
2170 V(I)=(R(I)-Q)/A(I,I):NEXT:NEXT
2180 RETURN

READY.
```

EASY CHANGES

1. The program uses 999 as the flag number to terminate various input modes. This may cause a problem if your data

include 999. You can easily change the flag number by modifying the value of EF in line 160 to any value not needed in your data. To use 10101, for example, make this change:

160 EF = 10101

2. To allow fits of higher degrees of 9, set the value of MD appropriately:

170 MD = 10

However, it must be stressed that it can be unreliable to attempt high degree fits. Unless your data is well behaved (X and Y values close to 1), the program will often not produce accurate results if D is greater than 5 or so. This is because sums of powers of X and Y are calculated up to powers of 2*D. These various sums are several orders of magnitude different from each other. Errors result because of the numerous truncation and round-off operations involved in doing arithmetic with them. A practical limit for MD is 7. The absolute limit of MD is 14.

MAIN ROUTINES

150- 170	Initializes constants.
200- 220	Dimensions arrays.
300- 380	Displays introductory messages.
400- 470	Gets X-Y input data from the user.
500- 550	Gets degree of polynomial from the user, determines if it is acceptable.
600- 670	Sets up equations for the simultaneous equation solver and calls it.
700- 780	Calculates goodness of fit, displays all results.
800- 860	Gets user's continuation option and branches to it.
900- 940	Determines Y value corresponding to any X value.
2000-2180	Subroutine to solve simultaneous linear algebraic equations.

MAIN VARIABLES

MX	Maximum number of data pairs allowed.
MD	Maximum degree allowed to fit.
EF	Ending flag value for data input and X point mode.

X,Y	Arrays of X and Y data points.
NP	Number of data pairs entered.
D	Degree of polynomial to fit.
D2	2*D, the maximum power sum to compute.
N	D + 1, number of simultaneous equations to solve.
A,R,V	Arrays for simultaneous linear equation solver.
P	Array for holding sums of various powers of X.
I,J,K,L	Loop indices.
Q,G	Work variables.
M	Mean value of Y.
T	Percent goodness of fit.
XV	Specific X point to calculate Y for.
YV	Y value corresponding to XV.

SUGGESTED PROJECTS

1. No provision for modifying the data is incorporated into the program. Often it would be nice to add, subtract, or modify parts of the data after some results are seen. Build in a capability to do this.
2. You may desire other forms of output. A useful table for many applications might include the actual X values, calculated Y values, and/or percentage errors in Y.
3. Sometimes certain points (or certain regions of points) are known to be more accurate than others. Then you would like to weight these points as being more important than others to be fit correctly. The least squares method can be modified to include such a weighting parameter with each data pair. Research this technique and incorporate it into the program. (Note: you can achieve some weighting with the current program by entering important points two or more times. There is a certain danger in this, however. You must only ask for a solution with D less than the number of *unique* data points. A division by zero error may result otherwise.)
4. Often you wish to try successively higher degree polynomials until a certain minimum goodness of fit is obtained. Modify the program to accept a minimally satisfactory goodness of fit from the user. Then have the program automatically try various polynomial fits until it finds the lowest degree fit, if any, with a satisfactory goodness of fit.

DIFFEQN

PURPOSE

Differential equations express functions by giving the rate of change of one variable with respect to another. This type of relation occurs regularly in almost all the physical sciences. The solution of these equations is necessary in many practical engineering problems.

For many such equations, a closed form (or exact analytical expression) solution can be obtained. However, for just as many, no such "simple" solution exists. The equation must then be solved numerically, usually by a computer program such as this.

There are many types and classes of differential equations. This program solves those of a simple type; namely, first order, ordinary differential equations. This means the equation to be solved can be written in the form

$$\frac{dY}{dX} = \text{(any function of X,Y)}$$

Here, X is the independent variable and Y is the dependent variable. The equation expresses the derivative (or rate of change) of Y with respect to X. The right-hand-side is an expression which may involve X and/or Y.

To use the program, you must supply it with the differential equation to be solved. The procedure to do this is explained in the "How To Use It" section.

A technique known as the "fourth-order, Runge-Kutta" method is used to solve the equation. Space limitations prevent

any detailed explanation of it here. However, it is discussed well in the numerical analysis books referenced in the bibliography.

HOW TO USE IT

The first thing you must do is enter the differential equation into the program. This must be done at line 3000. Currently this line contains a REM statement which you must replace. The form of line 3000 should be:

3000 D=(your function of X,Y)

D represents dY/dX. GOSUBs are made to line 3000 with X and Y set to their current values. Thus, when each RETURN is made, D will be set to the appropriate value of dY/dX for that given X and Y. If necessary, you may use the lines between 3000 and 3999 to complete the definition of D. Line 3999 already contains a RETURN statement so you do not need to add another one.

The program begins by warning you that you should have already entered the equation at line 3000. You acknowledge that this has been done by hitting the **C** key to continue.

Now the various initial conditions are input. You are prompted for them one at a time. They consist of: the initial values of X and Y, the stepsize interval in X at which to display the output, and the final value of X.

With the input phase completed, the program initializes things to begin the output. A question mark will be displayed in the lower left of the screen, telling you the program is waiting for you to hit any key to begin the output.

The two-column output is displayed at each interval of the stepsize until the final value of X is reached. Output may temporarily be halted at any time by simply hitting any key. This will stop the display until you hit any key to resume the output. The output may be started and stopped as often as desired, thus enabling you to leisurely view intermediate results before they scroll off the screen.

SAMPLE PROBLEM AND RUN

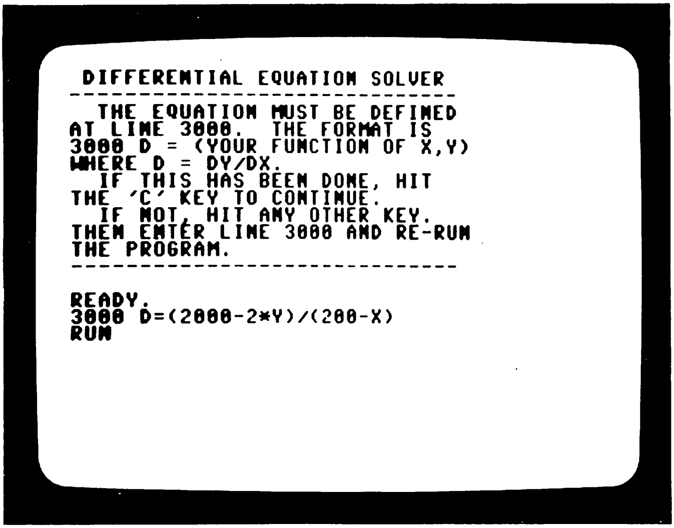
Problem: A body, originally at rest, is subjected to a force of 2000 dynes. Its initial mass is 200 grams. However, while it moves, it loses mass at the rate of 1 gram/sec. There is also an

air resistance equal to twice its velocity retarding its movement. The differential equation expressing this motion is:

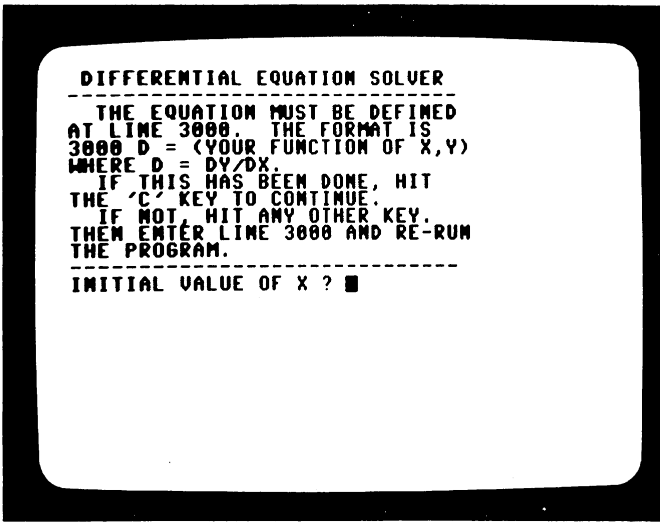
$$\frac{dY}{dX} = \frac{(2000 - 2Y)}{(200 - X)} \quad \text{where } Y = \text{velocity (cm./sec.)}, \quad X = \text{time (sec.)}$$

Find the velocity of the body every 10 seconds up through two minutes. Also, plot this velocity as a function of time.

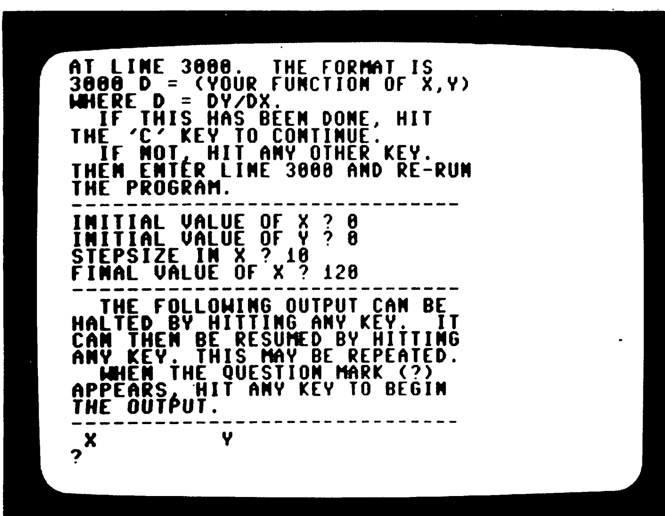
Solution and Sample Run: The solution and sample run are illustrated in the accompanying photographs.



The operator hits a key to exit from the program. Then he enters the differential equation into line 3000. He types RUN to restart the program.



The operator has hit the C key. The program responds by beginning the input phase.



The operator has completed the input. The program signals with a question mark that it is waiting for him to hit any key. It will not continue the run until he does so.

```
CAN THEN BE RESUMED BY HITTING
ANY KEY. THIS MAY BE REPEATED.
  WHEN THE QUESTION MARK (?)
APPEARS, HIT ANY KEY TO BEGIN
THE OUTPUT.
------------------------------
 X          Y
?
 0          0
 10         97.4999135
 20         189.999821
 30         277.49972
 40         359.999609
 50         437.499483
 60         509.999337
 70         577.499165
 80         639.998955
 90         697.498692
 100        749.998352
 110        797.497896
 120        839.997256

READY.
```

The operator hits a key and the program responds with the tabulated output. X is time in seconds and Y is velocity in cm./sec.

PROGRAM LISTING

```
READY.

100 REM: DIFFEQN
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 CLR
160 PRINT CHR$(147)
200 PRINT" DIFFERENTIAL ";
210 PRINT"EQUATION SOLVER"
220 GOSUB 2500:PRINT
230 PRINT"  THE EQUATION MUST ";
240 PRINT"BE DEFINED"
250 PRINT"AT LINE 3000.  THE ";
260 PRINT"FORMAT IS"
270 PRINT"3000 D = (YOUR ";
280 PRINT"FUNCTION OF X,Y)"
290 PRINT"WHERE D = DY/DX."
300 PRINT"  IF THIS HAS BEEN ";
310 PRINT"DONE, HIT"
320 PRINT"THE 'C' KEY TO ";
330 PRINT"CONTINUE."
```

```
340 PRINT"  IF NOT, HIT ANY ";
350 PRINT"OTHER KEY."
360 PRINT"THEN ENTER LINE 3000";
370 PRINT" AND RE-RUN"
380 PRINT"THE PROGRAM."
390 GOSUB 2500:PRINT
400 GET R$:IF R$ = "" THEN 400
410 IF R$ <> "C" THEN END
500 INPUT"INITIAL VALUE OF X ";XX
510 INPUT"INITIAL VALUE OF Y ";YY
520 Y = YY:X = XX:GOSUB 3000
530 INPUT"STEPSIZE IN X ";DX
540 INPUT"FINAL VALUE OF X ";XF
600 GOSUB 2500:PRINT
610 PRINT"  THE FOLLOWING ";
620 PRINT"OUTPUT CAN BE"
630 PRINT"HALTED BY HITTING ";
640 PRINT"ANY KEY.  IT"
650 PRINT"CAN THEN BE RESUMED ";
660 PRINT"BY HITTING"
670 PRINT"ANY KEY. THIS MAY ";
680 PRINT"BE REPEATED."
690 PRINT"  WHEN THE QUESTION MARK (?)"
710 PRINT"APPEARS, HIT ANY KEY TO BEGIN"
730 PRINT"THE OUTPUT."
740 GOSUB 2500:PRINT
800 PRINT" X"," Y":PRINT"?"
810 GET R$:IF R$ = "" THEN 810
900 PRINT XX,YY:GOSUB 1600
910 Q = XX+DX
920 IF Q > XF+1.E-5 THEN END
930 X = XX:Y = YY:GOSUB 3000:K0 = D
940 X = XX+DX/2:Y = YY+K0*DX/2
950 GOSUB 3000:K1 = D:Y = YY+K1*DX/2
960 GOSUB 3000:K2 = D:X = XX+DX
970 Y = YY+K2*DX:GOSUB 3000:K3 = D
980 DY = DX*(K0+2*K1+2*K2+K3)/6
990 YY = YY+DY:XX = XX + DX:GOTO 900
1600 GET R$
1610 IF R$ = "" THEN RETURN
1620 GET R$
1630 IF R$ = "" THEN 1620
1640 RETURN
2500 FOR J = 1 TO 30:PRINT"-";
2510 NEXT:RETURN
2900 REM
2910 REM *** DEFINE THE DIFFERENTIAL
2920 REM *** EQUATION IN LINES
2930 REM *** 3000 TO 3999.
2940 REM
```

```
2950 REM *** MAKE LINE 3000 THE FIRST
2960 REM *** LINE OF THE DEFINITION.
2970 REM
3000 REM D = <THE FUNCTION OF X,Y>
3999 RETURN

READY.
```

EASY CHANGES

1. If you have already entered the differential equation and wish to skip the introductory output, add this line:

   190 GOTO 500

   This will immediately begin the input dialog.
2. If you wish to use negative stepsizes, line 920 must be changed to:

   920 IF Q<XF − 1.E − 5 THEN END

MAIN ROUTINES

200- 390	Displays initial messages.
400- 540	Gets user's inputs.
600- 740	Displays additional messages.
800- 830	Initializes output display.
900- 990	Computes each step.
1600-1640	Stops and starts output.
2500-2510	Subroutine to display a dashed line.
3000-3999	User supplied routine to define D.

MAIN VARIABLES

D	Value of dY/dX.
X,Y	Values of X,Y on current step.
XX,YY	Values of X,Y on last step.
DX	Stepsize in X.
XF	Final value of X.
K0,K1, K2,K3	Runge-Kutta coefficients.
R$	User entered string.
Q	Work variable.
J	Loop index.

SUGGESTED PROJECTS

1. Modify the program to display the output in graphical form.
2. The value of dY/dX as a function of X is often a useful quantity to know. Modify the program to add it to the columnar display.
3. The inherent error in the calculation depends on the stepsize chosen. Most cases should be run with different stepsizes to insure the errors are not large. If the answers do not change much, you can be reasonably certain that your solutions are accurate. Better yet, techniques exist to vary the stepsize during the calculation to insure the error is sufficiently small during each step. Research these methods and incorporate them into the program.
4. The program can be easily broadened to solve a set of coupled, first order, differential equations simultaneously. This would greatly increase the types of problems that could be solved. Research this procedure and expand the program to handle it.

GRAPH

PURPOSE

Is a picture worth a thousand words? In the case of mathematical functions, the answer is often "yes." A picture, i.e. a graph, enables you to see the important behavior of a function quickly and accurately. Trends, minima, maxima, etc. become easy and convenient to determine.

GRAPH produces a two-dimensional color plot of a function that you supply. The function must be in the form Y=(any function of X). The independent variable X will be plotted along the abscissa (horizontal axis). The dependent variable Y will be plotted along the ordinate (vertical axis). You have complete control over the scaling that is used on the X and Y axes.

HOW TO USE IT

Before running the program, you must enter into it the function to be plotted. This is done as a subroutine beginning at line 5000. It must define Y as a function of X. The subroutine will be called with X set to various values. It must then set the variable Y to the correct corresponding value. The subroutine may be as simple or as complex as necessary to define the function. It can take one line or several hundred lines. Line 5999 is already set as a RETURN statement, so you need not add another one.

Having entered this subroutine, you are ready to run the program. The program begins by warning you that it assumes the function has already been entered at line 5000. It will then ask you for the domain of X, i.e. the lowest and highest values of X

that you wish to have plotted. Values can be positive or negative as long as the highest value is actually larger than the lowest one.

Now you must choose the scale for Y. To do this intelligently, you probably need to know the minimum and maximum values of Y over the domain of X selected. The program finds these values and displays them for you. You must then choose the minimum and maximum values you wish to have on the Y scale. Again, any two values are acceptable as long as the maximum scale value of Y is larger than the minimum scale value of Y.

The program will now display the plot of your function. Each axis is twenty characters long, with the origin defined as the minimum scale values of both X and Y. Ten tick marks appear on each axis. The locations of the lower, middle, and upper values on each scale are displayed appropriately. Note: The program may not be able to display all six scaling numbers if you input some scaling having a large number of significant digits. If this occurs on the X axis, the program may use one or more of the expressions "XL", "XM" or "XU" instead of the actual numbers. This stands respectively for "Xlower", "Xmid" and "Xupper". Similarly, "YL", "YM" and "YU" may be used on the Y axis.

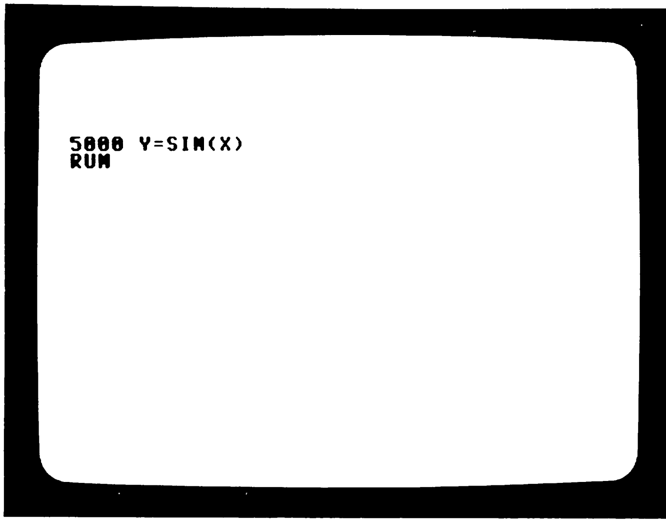
The actual plot is drawn with twice the resolution shown on the axis. That is, forty values of X and Y are plotted. This is accomplished by using the various 2×2 graphic characters available on the Commodore 64.

If a value for Y should be off-scale, a special orange colored point is displayed at the appropriate value of X. If the actual value of Y is too large, it is plotted just above the maximum Y value. If this actual value of Y is too small, it is plotted just below the Y axis.

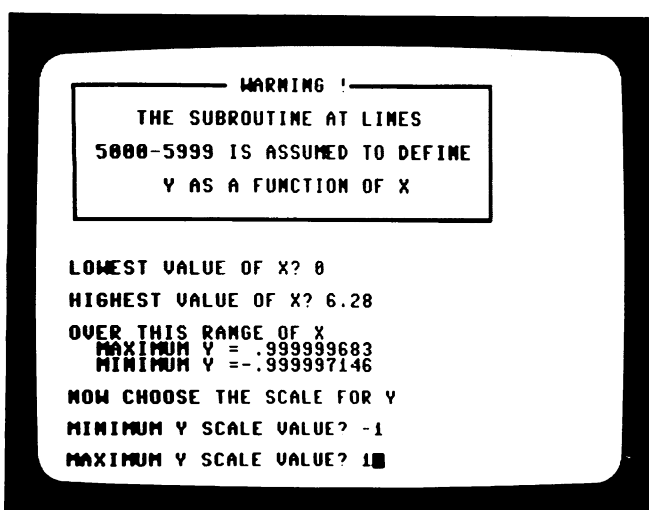
After the plot is drawn, the program will tell you to hit any key to continue. When you do so, information about the plot scaling is provided. For both X and Y, you are given the low, mid, and upper values on each axis.

You now have the option of hitting G to draw the graph again or any other key to terminate the program.

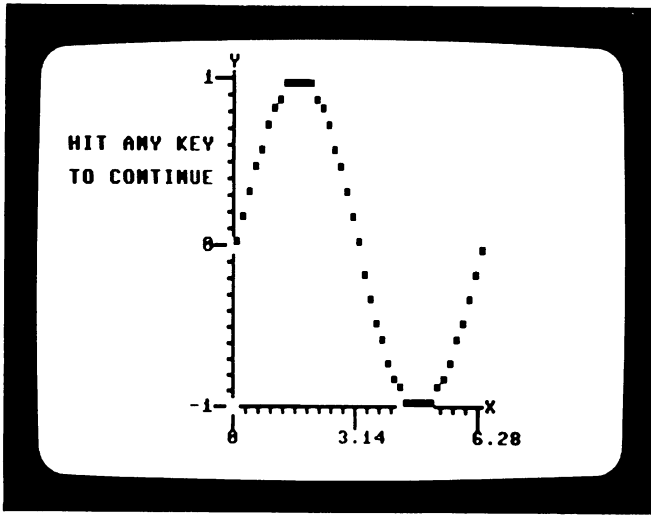
SAMPLE RUN



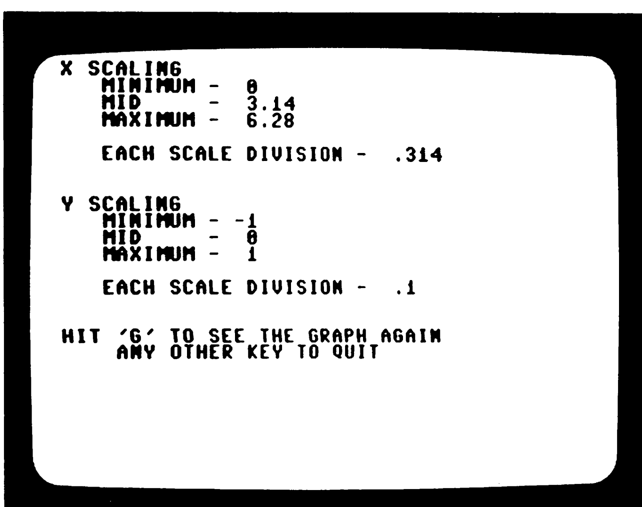
After loading the program, the operator enters line 5000 to request the graph Y = SIN(X). RUN is typed to begin the program.



The input dialog transpires. The operator asks that the domain of X be 0-6.28. The program responds by showing the maximum and minimum value of Y over this domain. The operator chooses an appropriate scale for the Y axis.



The graph is displayed as requested. The program waits for the operator to hit any key to continue.



Relevant scaling information is shown. By pressing G, the operator can see the graph again.

PROGRAM LISTING

```
READY.

100 REM: GRAPH
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 C=1876:XV=12:POKE 53280,1:POKE 53281,0
160 PB=102:C1=56148:C0=1:C2=5:C3=8
170 W=221
200 POKE 53281,1:PRINT CHR$(147);:POKE 53281,0
201 PRINT TAB(16);CHR$(18);"GRAPH":PRINT
210 PRINT:GOSUB 1000
220 PRINT:PRINT:INPUT"LOWEST VALUE OF X";XL
225 PRINT:INPUT"HIGHEST VALUE OF X";XU
230 IF XU<=XL THEN PRINT:PRINT"---BAD X RANGE---"
235 IF XU<=XL THEN GOTO 220
240 GOSUB 800:GOSUB 300:GOSUB 500
245 PRINT CHR$(19):PRINT:PRINT:PRINT:PRINT
250 PRINT"HIT ANY KEY":PRINT:PRINT"TO CONTINUE"
260 GET Q$:IF Q$="" THEN 260
270 GOSUB 900
280 END
300 POKE 53281,1:PRINT CHR$(147);
301 POKE 53281,0:POKE C,91
310 FOR J=1 TO 20:POKE C+J,114
320 POKE C-40*J,115:NEXT
330 FOR J=0 TO 20 STEP 10
340 POKE C+40+J,93:POKE C-1-40*J,64
350 NEXT:POKE C+21,24:POKE C-840,25
360 XM=(XL+XU)/2:YM=(YL+YU)/2
370 A$=STR$(XL):A=LEN(A$):TB=XV-A+1
375 IF TB<1 THEN TB=XV:A$="XL"
380 GOSUB 460:A$=STR$(XM):A=LEN(A$):TB=XV+10-A/2+1
385 GOSUB 460:A$=STR$(XU):TB=XV+19
390 IF TB+LEN(A$)>39 THEN TB=XV+20:A$="XU"
400 GOSUB 460:A$=STR$(YU):A=LEN(A$):TB=XV-A-1
405 IF TB<0 THEN TB=XV-3:A$="YU"
410 NL=1:GOSUB 480
415 A$=STR$(YM):A=LEN(A$):TB=XV-A-1:NL=11
420 IF TB<0 THEN TB=XV-3:A$="YM"
430 GOSUB 480:A$=STR$(YL):A=LEN(A$):TB=XV-A-1
435 IF TB<0 THEN TB=XV-3:A$="YL"
440 NL=21:GOSUB 480:RETURN
460 PRINT CHR$(19)
463 FOR J=1 TO 22:PRINT CHR$(17);:NEXT
466 PRINT TAB(TB);A$:RETURN
```

```
480 PRINT CHR$(19);
483 FOR J=1 TO NL:PRINT CHR$(17);:NEXT
486 PRINT TAB(TB);A$:RETURN
500 DX=(XU-XL)/20:DY=(YU-YL)/20
510 X=XL:GOSUB 700:PC=124:IF F=0 THEN PC=108
520 IF D<0 THEN D=0:PC=PB
530 IF D>20 THEN D=20:PC=PB
532 C0=C2:IF PC = PB THEN C0 = C3
535 POKE C1-40*D,C0
540 POKE C-40*D,PC
550 FOR J=1 TO 20:X=XL+DX*(J-0.5):GOSUB 700
555 PC=126:IF F=0 THEN PC=123
560 IF D<0 THEN D=0:PC=PB
570 IF D>20 THEN D=20:PC=PB
575 C0=C2:IF PC = PB THEN C0 = C3
580 POKE C1+J-40*D,C0
585 POKE C+J-40*D,PC
590 X=XL+DX*J:GOSUB 700:PK=PEEK(C+J-40*D)
595 IF D<0 THEN D=0:PC=PB:GOTO 660
600 IF D>20 THEN D=20:PC=PB:GOTO 660
610 PC=226:IF F=0 THEN PC=127
620 IF PK=126 THEN 660
630 PC=255:IF F=0 THEN PC=98
640 IF PK=123 THEN 660
650 PC=124:IF F=0 THEN PC=108
655 C0=C2:IF PC = PB THEN C0 = C3
660 POKE C1+J-40*D,C0
665 POKE C+J-40*D,PC
670 NEXT:RETURN
700 GOSUB 5000:V=(Y-YL)/DY:D=INT(V)
710 IF Y<YL THEN D=-5
720 IF Y>YU THEN D=25
730 F=1:IF (V-D)>=0.5 THEN D=D+1:F=0
740 RETURN
800 DX=(XU-XL)/40:X=XL:GOSUB 5000:MN=Y:MX=Y
805 FOR J=1 TO 40:X=XL+J*DX:GOSUB 5000
810 IF Y>MX THEN MX=Y
820 IF Y<MN THEN MN=Y
830 NEXT
840 PRINT:PRINT"OVER THIS RANGE OF X"
845 PRINT"  MAXIMUM Y =";MX
850 PRINT"  MINIMUM Y =";MN:PRINT
855 PRINT"NOW CHOOSE THE SCALE FOR Y":PRINT
860 INPUT"MINIMUM Y SCALE VALUE";YL
865 PRINT:INPUT"MAXIMUM Y SCALE VALUE";YU
870 IF YU<=YL THEN PRINT
873 IF YU<=YL THEN PRINT"--- BAD Y SCALING---"
876 IF YU<=YL THEN GOTO 840
880 RETURN
900 PRINT CHR$(147);"X SCALING"
```

```
910 PRINT"   MINIMUM - ";XL
913 PRINT"   MID     - ";XM
916 PRINT"   MAXIMUM - ";XU
920 PRINT:PRINT"   EACH SCALE DIVISION - ";DX
925 PRINT:PRINT:PRINT"Y SCALING"
930 PRINT"   MINIMUM - ";YL
933 PRINT"   MID     - ";YM
936 PRINT"   MAXIMUM - ";YU
940 PRINT:PRINT"   EACH SCALE DIVISION - ";DY
945 PRINT:PRINT
950 PRINT"HIT 'G' TO SEE THE GRAPH AGAIN"
955 PRINT"    ANY OTHER KEY TO QUIT
960 GET Q$:IF Q$="" THEN 960
970 IF Q$="G" THEN GOSUB 300:GOSUB 500
980 RETURN
1000 PRINT CHR$(176);:FOR J = 1 TO 11
1010 PRINT CHR$(192);:NEXT:PRINT" WARNING !";
1015 FOR J = 1 TO 11
1020 PRINT CHR$(192);:NEXT
1025 PRINT CHR$(174):GOSUB 1100:PRINT CHR$(W);
1030 PRINT"    THE SUBROUTINE AT LINES     ";
1035 PRINT CHR$(W):GOSUB 1100
1040 PRINT CHR$(W);
1043 PRINT " 5000-5999 IS ASSUMED TO DEFINE ";
1046 PRINT CHR$(W):GOSUB 1100
1050 PRINT CHR$(W);
1053 PRINT "       Y AS A FUNCTION OF X      ";
1056 PRINT CHR$(W):GOSUB 1100
1060 PRINT CHR$(173);:FOR J = 1 TO 32
1070 PRINT CHR$(192);:NEXT:PRINT CHR$(189)
1080 RETURN
1100 PRINT CHR$(W);SPC(32);CHR$(W):RETURN
4970 REM
4980 REM SUBROUTINE AT 5000 MUST BE SET
4990 REM
5000 REM *** Y=F(X) GOES HERE
5999 RETURN

READY.
```

EASY CHANGES

1. You may want the program to self-scale the Y axis for you. That is, you want it to use the minimum and maximum Y values that it finds as the limits on the Y axis. This can be accomplished by adding the following line:

   835 YU=MX:YL=MN:RETURN

2. Do you sometimes forget to enter the subroutine at line 5000 despite the introductory warning? As is, the program will plot the straight line Y = 0 if you do this. If you want a more drastic reaction to prevent this, change line 5000 to read

   5000 Y = 1/0

   Now, if you don't enter the actual subroutine desired, the program will stop and print the following message after you enter the X scaling values.

   ?DIVISION BY ZERO ERROR IN 5000

MAIN ROUTINES

150- 170	Initializes constants.
200- 210	Displays introductory warning.
220- 280	Mainline routine—gets X scaling from user and calls various subroutines.
300- 486	Subroutines to draw graph axes and scale labeling.
500- 670	Subroutine to plot the function.
700- 740	Subroutine to determine the plotting position for Y.
800- 880	Subroutine which determines the minimum, maximum Y values; gets Y scale from user.
900- 980	Subroutine which displays the scaling parameters, asks user if he wants the graph re-plotted.
1000-1100	Subroutines to display the introductory warning.
5000-5999	User supplied subroutine to evaluate Y as a function of X.

MAIN VARIABLES

C1	Base Poke argument for color setting.
C0	Color for current graph point.
C2	Color for normal graph point.
C3	Color for error graph point.
XL,XM, XU	Lower, middle, upper scale values of X.
YL,YM, YU	Lower, middle, upper scale values of Y.
DX,DY	Scale increments of X,Y.
X,Y	Current values of X,Y.

C	Poke argument for the plot origin.
PC	Poke argument for normal plotting.
PB	Poke argument for off-axis plotting.
PK	Peek value.
W	CHR$ argument.
TB,XV	Tab arguments.
F	Y position flag (0 = down, 1 = up).
V	Value of X or Y in scale units.
D	Integer value of V.
MN,MX	Minimum, maximum values of Y.
A$	String representation of axis numbers.
A	Length of A$.
NL	Number of lines to print.
Q$	User reply string.
J	Loop index.

SUGGESTED PROJECTS

1. Determine and display the values of X at which the minimum and maximum values of Y occur.
2. After the graph is plotted, allow the user to obtain the exact value of Y for any given X.
3. Expand the graph to a 30 character width in the X direction.

INTEGRATE

PURPOSE AND DEFINITION

The need to evaluate integrals occurs frequently in much scientific and mathematical work. This program will numerically integrate a function that you supply using a technique known as Simpson's rule. It will continue to grind out successive approximations of the integral until you are satisfied with the accuracy of the solution.

Mathematical integration will probably be a familiary term to those who have studied some higher mathematics. It is a fundamental subject of second-year calculus. The integral of a function between the limits $x = l$ (lower limit) and $x = u$ (upper limit) represents the area under its curve; i.e. the shaded area in Figure 1.

We may approximate the integral by first dividing up the area into rectangular strips or segments. We can get a good estimate of the total integral by summing the areas of these segments by using a parabolic fit across the top. For those who understand some mathematical theory, Simpson's rule may be expressed as

$$\int_{x=l}^{x=u} f(x)dx \cong \frac{\Delta}{3} \left\{ f(l) + f(u) + 4 \sum_{j=1}^{N/2} f[l + \Delta(2j-1)] + 2 \sum_{j=1}^{(N-2)/2} f[l + 2\Delta j] \right\}$$

Here N is the number of segments into which the total interval is divided. N is 4 in the diagram.

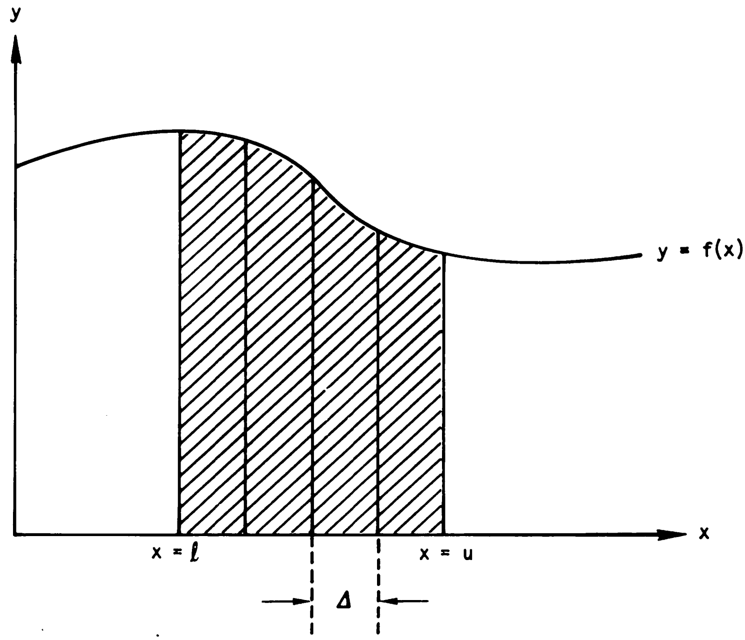


Figure 1. The Integral of f(x).

For a good discussion of the numerical evaluation of integrals see: McCracken, Dorn, *Numerical Methods and Fortran Programming*, New York, Wiley, 1964, pp. 160. Don't let the word "Fortran" scare you away. The discussions in the book are independent of programming language with only some program examples written in Fortran.

HOW TO USE IT

The program begins with a warning! This is to remind you that you should have already entered the subroutine to evaluate Y as a function of X. This subroutine must start at line 7000. More about it shortly.

You will then be asked to provide the lower and upper limits of the integration domain. Any numerical values are acceptable. It is not even necessary that the lower limit of X be smaller than the upper one.

The program will now begin displaying its numerical evaluations of the integral. The number of segments used in the calculation continually doubles. This causes the accuracy of the integral to increase at the expense of additional computation time. For most functions, you should see the value of the integral converging quickly to a constant (or near constant) value. This, of course, will be the best numerical evaluation of the integral at hand.

When you are satisfied with the accuracy of the solution, you must hit the **RUN/STOP** key to terminate the program. If not, the program will run forever (assuming you can pay the electric bills). The amount of computation is approximately doubled each step. This means it will take the computer about the same amount of time to compute the next step that it took to compute *all* the previous steps. Thus, it will soon be taking the Commodore 64 hours, days, and weeks to compute steps. Eventually, round-off errors begin degrading the results, causing a nice, constant, converged solution to change. However, the high precision of the computer's floating point arithmetic will postpone this for quite a while. You will probably lose patience before seeing it.

The function to be integrated can be as simple or as complicated as you desire. It may take one line or a few hundred lines of code. In any case, the subroutine to express it must start at line 7000. This subroutine will be continually called with the variable X set. When it returns, it should have set the variable Y to the corresponding value of the function for the given X. The subroutine must be able to evaluate the function at any value of X between the lower and upper bounds of the integration domain.

If your function consists of experimental data at discrete values of X, you must do something to enable the subroutine to evaluate the function at intermediate values of X. We recommend one of two approaches. First, you could write the subroutine to linearly interpolate the value of Y between the appropriate values of X. This will involve searching your data table for the pair of experimental X values that bound the value of X where the function is to be evaluated. Secondly, the program CURVE presented elsewhere in this section can produce an approximate polynomial expression to fit your experimental data. This expression can then be easily entered as the subroutine at line 7000.

By the way, Simpson's rule is *exact* for any polynomial of degree 3 or less. This means that if the function can be written in the form

$$Y=A*X*X*X+B*X*X+C*X+D$$

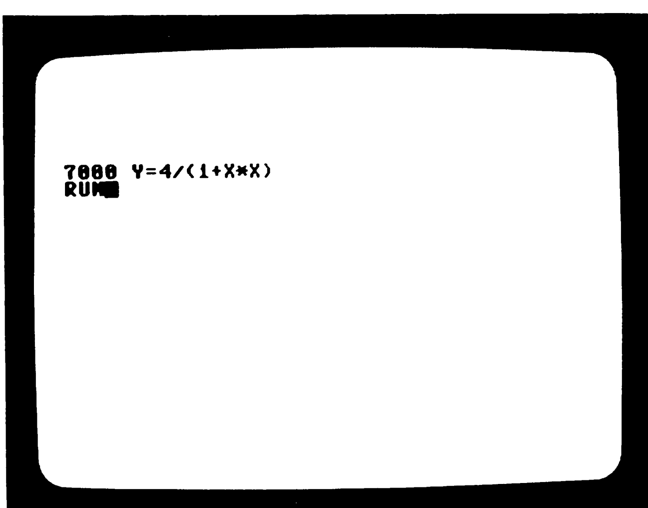
where A, B, C, D are constants, the program will calculate the integral exactly even with only two segments.

SAMPLE RUN

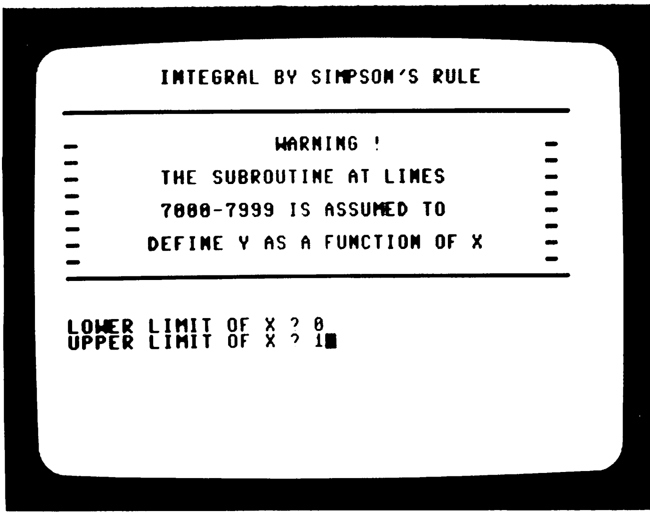
The sample run illustrates the following integration

$$\int_{x=0}^{x=1} \frac{4}{1+x^2}\,dx$$

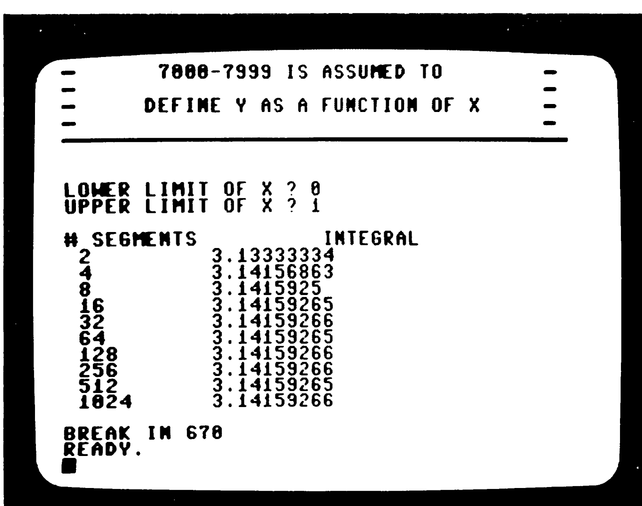
This integral has the theoretical value of π (pi) as the correct answer! Pi, as you may know, has the value 3.1415926535. . . . Before the run is started, the above function is entered at line 7000.



The operator enters the integrand function at line 7000 and types RUN to start the program.



The upper and lower bounds of X are entered as requested.



The results are computed up to 1024 segments. Then the RUN/STOP key is pressed to terminate the calculation.

PROGRAM LISTING

```
READY.

100 REM: INTEGRATE
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 CLR
160 N = 2
200 PRINT CHR$(147)
210 C$ = CHR$(192)
220 PRINT"          INTEGRAL BY SIMPSON'S RULE"
230 PRINT:GOSUB 400:GOSUB 420
240 PRINT"                   WARNING !";
250 GOSUB 430:GOSUB 440
260 GOSUB 420
270 PRINT"         THE SUBROUTINE AT";
280 PRINT" LINES";:GOSUB 430
290 GOSUB 440:GOSUB 420
300 PRINT"         7000-7999 IS";
310 PRINT" ASSUMED TO";
320 GOSUB 430:GOSUB 440
330 GOSUB 420:PRINT"       DEFINE Y";
340 PRINT" AS A FUNCTION OF X";
350 GOSUB 430:GOSUB 440
360 GOSUB 400:PRINT:GOTO 500
400 FOR J = 1 TO 39
410 PRINT C$;:NEXT:PRINT C$:RETURN
420 PRINT C$;:RETURN
430 PRINT TAB(38);C$:RETURN
440 GOSUB 420:GOSUB 430:RETURN
500 INPUT"LOWER LIMIT OF X ";L
510 INPUT"UPPER LIMIT OF X ";U
550 PRINT
560 PRINT"# SEGMENTS","INTEGRAL"
600 DX = (U-L)/N:T = 0
610 X = L:GOSUB 7000:T = T+Y
620 X = U:GOSUB 7000:T = T+Y
650 M = N/2:Z = 0
660 FOR J = 1 TO M
670 X = L+DX*(2*J-1):GOSUB 7000
680 Z = Z+Y:NEXT:T = T+4*Z
700 M = M-1:IF M = 0 THEN 800
710 Z = 0:FOR J = 1 TO M
720 X = L+DX*2*J:GOSUB 7000:Z = Z+Y
730 NEXT:T = T+2*Z
800 A = DX*T/3
810 PRINT N,A
820 N = N*2
```

```
830 GOTO 600
6960 REM
6970 REM *** ENTER SUBROUTINE AT
6980 REM *** LINE 7000
6990 REM
7000 REM *** Y = F(X) GOES HERE
7999 RETURN

READY.
```

EASY CHANGES

1. You might want the program to stop calculation after the integral has been evaluated for a given number of segments. Adding the following line will cause the program to stop after the integral is evaluated for a number of segments greater than or equal to 100.

   815 IF N> =100 THEN END

   Of course, you may use any value you wish instead of 100.
2. Perhaps you would like to see the number of segments change at a different rate during the course of the calculation. This can be done by modifying line 820. To increase the rate of change, try

   820 N=N*4

   to change it at a constant (and slower) rate, try

   820 N=N+50

   Be sure, however, that the value of N is always even.
3. You can experiment with the border used around the introductory warning by changing the CHR$ argument used in line 210. This might be particularly desirable if you are using a black and white TV. To get a pleasing asterisk border, which looks good on a black and white set, try

   210 C$=CHR$(42)

MAIN ROUTINES

150- 160	Initializes constants.
200- 360	Displays introductory messages and warning.
400- 440	Graphics display subroutines.
500- 510	Gets integration limits from operator.
550- 560	Displays column headings.

600- 620	Computes integral contribution from end points.
650- 680	Adds contribution from one summation.
700- 730	Adds contribution from other summation.
800- 830	Completes integral calculation and displays it. Increases number of segments and restarts calculation.
7000-7999	Operator supplied subroutine to evaluate f(x).

MAIN VARIABLES

N	Number of segments.
J	Loop index.
L,U	Lower, Upper integration limit of x.
DX	Width of one segment.
T	Partial result of integral.
M	Number of summations.
Z	Subtotal of summations.
A	Value of integral.
X	Current value of x.
Y	Current value of the function y = f(x).
C$	String used in messages.

SUGGESTED PROJECTS

1. Research other similar techniques for numerical integration such as the simpler trapezoid rule. Then compute the integral with this new method. Compare how the two methods converge toward the (hopefully) correct answer.

SIMEQN

PURPOSE

This program solves a set of simultaneous linear algebraic equations. This type of problem often arises in scientific and numerical work. Algebra students encounter them regularly—many "word" problems can be solved by constructing the proper set of simultaneous equations.

The equations to be solved can be written mathematically as follows:

$$
\begin{array}{l}
A_{11}X_1 + A_{12}X_2 + \ldots + A_{1N}X_N = R_1 \\
A_{21}X_1 + A_{22}X_2 + \ldots + A_{2N}X_N = R_2 \\
\quad \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \qquad\quad \vdots \\
A_{N1}X_1 + A_{N2}X_2 + \ldots + A_{NN}X_N = R_N
\end{array}
$$

N is the number of equations and thus the number of unknowns also. The unknowns are denoted X_1 through X_N.

Each equation contains a coefficient multiplier for each unknown and a right-hand-side term. These coefficients (the A matrix) and the right-hand-sides (R_1 through R_N) must be constants—positive, negative, or zero. The A matrix is denoted with doubled subscripts. The first subscript is the equation number and the second one is the unknown that the coefficient multiplies.

HOW TO USE IT

The program will prompt you for all necessary inputs. First, it asks how many equations (and thus how many unknowns) com-

prise your set. This number must be at least 1. If it is too large, an OUT OF MEMORY or BAD SUBSCRIPT error will immediately result.

Next, you must enter the coefficients and right-hand-sides for each equation. The program will request these one at a time, continually indicating which term it is expecting next.

Once it has all your inputs, the program begins calculating the solution. This may take a little while if the value of N is high. The program ends by displaying the answers. These, of course, are the values of each of the unknowns, X_1 through X_N.

If you are interested, the numerical technique used to solve the equations is known as Gaussian elimination. Row interchange to achieve pivotal condensation is employed. (This keeps maximum significance in the numbers.) Then back substitution is used to arrive at the final results. This technique is much simpler than it sounds and is described well in the numerical analysis books referenced in the bibliography.

SAMPLE PROBLEM AND RUN

Problem: A painter has a large supply of three different colors of paint: dark green, light green, and pure blue. The dark green is 30% blue pigment, 20% yellow pigment, and the rest base. The light green is 10% blue pigment, 35% yellow pigment, and the rest base. The pure blue is 90% blue pigment, no yellow pigment, and the rest base. The painter, however, needs a medium green to be composed of 25% blue pigment, 25% yellow pigment, and the rest base. In what percentages should he mix his three paints to achieve this mixture?

Solution: Let $X_1 =$ percent of dark green to use,
$X_2 =$ percent of light green to use,
$X_3 =$ percent of pure blue to use.

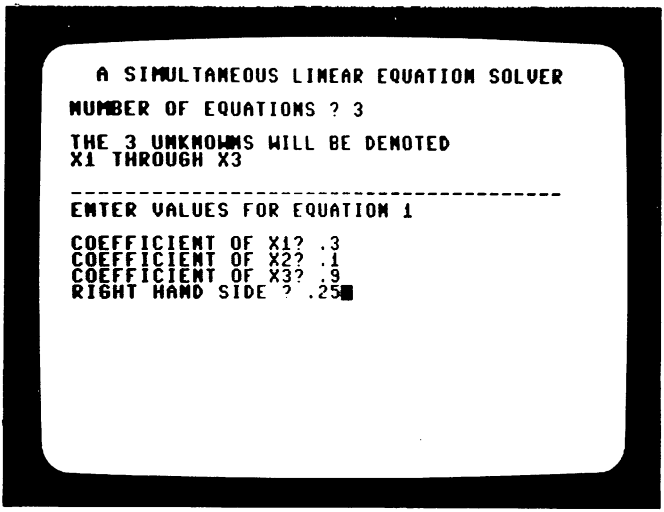
The problem leads to these three simultaneous equations to solve:

$$\begin{aligned} 0.3\,X_1 + 0.1\,X_2 + 0.9\,X_3 &= 0.25 \\ 0.2\,X_1 + 0.35\,X_2 &= 0.25 \\ X_1 + X_2 + X_3 &= 1.0 \end{aligned}$$

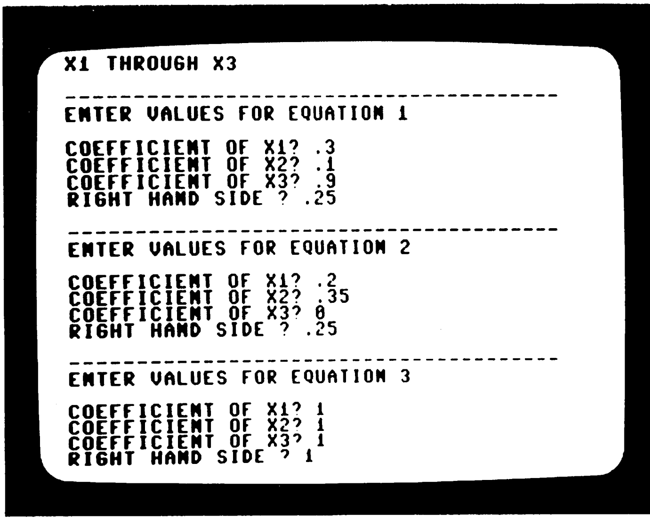
The first equation expresses the amount of blue pigment in the mixture. The second equation is for the yellow pigment. The third equation states that the mixture is composed entirely of the

three given paints. (Note that all percentages are expressed as numbers from 0-1.) The problem leads to the following use of SIMEQN.

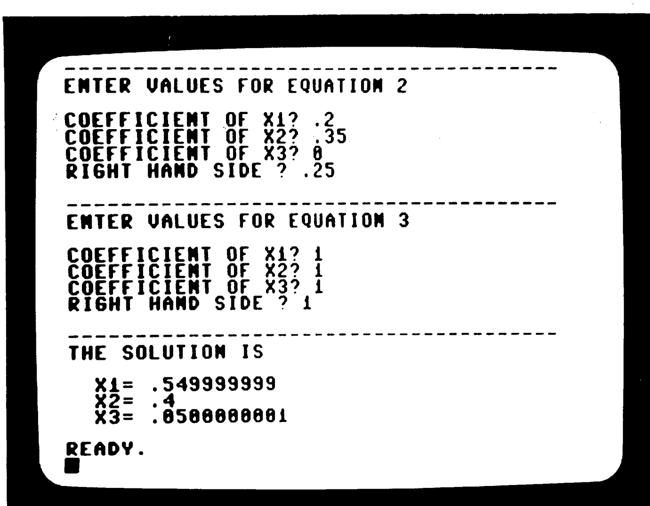
SAMPLE RUN



The operator chooses to solve a set of three simultaneous equations and then enters the coefficients for the first equations.



The coefficients for the remaining two equations are entered.



The computer provides the solution. Rounded to the nearest percent, the painter should use a mixture of 55% dark green, 40% light green, and 5% pure blue.

PROGRAM LISTING

```
READY.


100 REM: SIMEQN
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 CLR
200 PRINT CHR$(147)
210 PRINT"  A SIMULTANEOUS ";
220 PRINT"LINEAR EQUATION SOLVER"
240 PRINT
250 INPUT"NUMBER OF EQUATIONS ";N
260 N = INT(N):IF N > 0 THEN 400
270 PRINT:PRINT"** ERROR ! **"
280 PRINT"THERE MUST BE AT LEAST 1"
300 GOTO 240
400 DIM A(N,N),R(N),V(N)
410 PRINT
420 PRINT"THE";N;"UNKNOWNS WILL BE DENOTED"
430 PRINT"X1 THROUGH X";
440 PRINT MID$(STR$(N),2)
460 GOSUB 900:FOR J = 1 TO N
470 PRINT"ENTER VALUES FOR EQUATION";J
490 PRINT:FOR K = 1 TO N
500 PRINT"COEFFICIENT OF X";MID$(STR$(K),2);
520 INPUT A(J,K):NEXT
530 INPUT"RIGHT HAND SIDE ";R(J)
540 GOSUB 900:NEXT
550 GOSUB 2000
600 PRINT"THE SOLUTION IS"
610 PRINT:FOR J = 1 TO N
620 PRINT"  X";MID$(STR$(J),2);
630 PRINT"=";V(J)
640 NEXT:END
900 PRINT:FOR L = 1 TO 39
910 PRINT"-";:NEXT:PRINT:RETURN
2000 IF N > 1 THEN 2020
2010 V(1) = R(1)/A(1,1):RETURN
2020 FOR K = 1 TO N-1:I = K+1
2030 L = K
2040 Q = ABS(A(I,K))-ABS(A(L,K))
2050 IF Q = 0 THEN L = I
2060 IF I < N THEN I = I+1:GOTO 2040
2070 IF L = K THEN 2110
2080 FOR J = K TO N:Q = A(K,J)
2090 A(K,J) = A(L,J):A(L,J) = Q:NEXT
2100 Q = R(K):R(K) = R(L):R(L) = Q
2110 I = K+1
2120 Q = A(I,K)/A(K,K):A(I,K) = 0
2130 FOR J = K+1 TO N
```

```
2140 A(I,J) = A(I,J) - Q*A(K,J):NEXT
2150 R(I) = R(I) - Q*R(K)
2160 IF I < N THEN I = I+1:GOTO 2120
2170 NEXT
2180 V(N) = R(N)/A(N,N)
2190 FOR I = N-1 TO 1 STEP-1
2200 Q = 0:FOR J = I+1 TO N
2210 Q =Q+A(I,J)*V(J)
2220 V(I) = (R(I)-Q)/A(I,I):NEXT
2230 NEXT:RETURN

READY.
```

EASY CHANGES

You may be surprised sometime to see the program fail completely and display this message:

?DIVISION BY ZERO ERROR IN 2180

This means your input coefficients (the A array) were ill-conditioned and no solution was possible. This can arise from a variety of causes; e.g. if one equation is an exact multiple of another, or if *every* coefficient of one particular unknown is zero. If you would like the program to print a diagnostic message in these cases add these lines.

```
2172 IF A(N,N)< >0 THEN 2180
2174 PRINT "BAD INPUT –";
2176 PRINT "NO SOLUTION POSSIBLE"
2178 STOP
```

MAIN ROUTINES

200- 240	Clears screen and displays program title.
250- 550	Gets input from user and calculates the solution.
600- 640	Displays the solution.
900- 910	Subroutine to space and separate the output.
2000-2330	Subroutine to calculate the solution; consisting of the following parts:
2000-2010	Forms solution if N = 1.
2020-2170	Gaussian elimination.
2030-2110	Interchanges rows to achieve pivotal condensation.
2180-2230	Back substitution.

MAIN VARIABLES

I,J,K,L	Loop indices and subscripts.
N	Number of equations (thus number of unknowns also).
A	Doubly dimensioned array of the coefficients.
R	Array of right-hand-sides.
V	Array of the solution.
Q	Work variable.

SUGGESTED PROJECTS

1. The program modifies the A and R arrays while computing the answer. This means the original input cannot be displayed after it is input. Modify the program to save the information and enable the user to retrieve it after the solution is given.
2. Currently, a mistake in typing input cannot be corrected once the **RETURN** key is pressed after typing a number. Modify the program to allow correcting previous input.

STATS

PURPOSE

Ever think of yourself as a statistic? Many times we lament at how we have become just numbers in various computer memories, or we simply moan at our insurance premiums. To most people, the word "statistics" carries a negative connotation. To invoke statistics is almost to be deceitful, or at least dehumanizing. But really, we all use statistical ideas regularly. When we speak of things like "she was average height" or the "hottest weather in years," we are making observations in statistical terms. It is difficult not to encounter statistics in our lives, and this book is no exception.

Of course, when used properly, statistics can be a powerful, analytical tool. STATS analyzes a set of numerical data that you provide. It will compile your list, order it sequentially, and/or determine several statistical parameters which describe it.

This should prove useful in a wide variety of applications. Teachers might determine grades by analyzing a set of test scores. A businessman might determine marketing strategy by studying a list of sales to clients. Little leaguers always like to pore over the current batting and pitching averages. You can probably think of many other applications.

HOW TO USE IT

First, your data list must be entered. The program will prompt you for each value with a question mark. Two special inputs, *END and *BACK, may be used at any time during this data in-

put phase. To signal the end of data, input the four character string, *END, in response to the (last) question mark. You must, of course, enter at least one data value.

If you discover that you have made a mistake, the five character string, *BACK, can be used to back up the input process. This will cause the program to re-prompt you for the previous entry. By successive uses of *BACK you can return to any previous position.

With the input completed, the program enters a command mode. You have four options to continue the run:

1) List the data in the order input
2) List the data in ranking order
3) Display statistical parameters
4) End the program

Simply input the number 1, 2, 3, or 4 to indicate your choice. If one of the first three is selected, the program will perform the selected function and return to this command mode to allow another choice. This will continue until you choose 4 to terminate the run. A description of the various options now follows.

Options 1 and 2 provide lists of the data. Option 1 does it in the original input order while option 2 sorts the data from highest value to lowest.

The lists are started by hitting any key when told to do so. Either list may be temporarily halted by hitting any key while the list is being displayed. This allows you to leisurely view data that might otherwise start scrolling off the screen. Simply hit any key to resume the display. This starting and stopping can be repeated as often as desired. When the display is completed, you must again hit a key to re-enter the command mode.

Option 3 produces a statistical analysis of your data. Various statistical parameters are calculated and displayed. The following is an explanation of some that may not be familiar to you.

Three measures of location, or central tendency, are provided. These are indicators of an "average" value. The *mean* is the sum of the values divided by the number of values. If the values are arranged in order from highest to lowest, the *median* is the middle value if the number of values is odd. If it is even, the median is the number halfway between the two middle values. The *midrange* is the number halfway between the largest and smallest values.

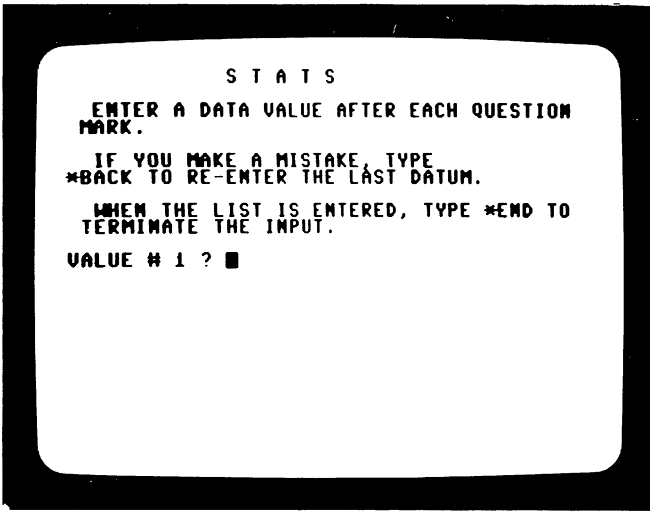
These measures of location give information about the average value of the data. However, they give no idea of how the data is dispersed or spread out around this "average." For that we need "measures of dispersion" or as they are sometimes called, "measures of variation." The simplest of these is the *range* which is just the difference between the highest and lowest data values. Two other closely related measures of dispersion are given: the *variance* and the *standard deviation*. The variance is defined as:

$$VA = \frac{\sum_{i=1}^{N} (V_i - M)^2}{N - 1}$$

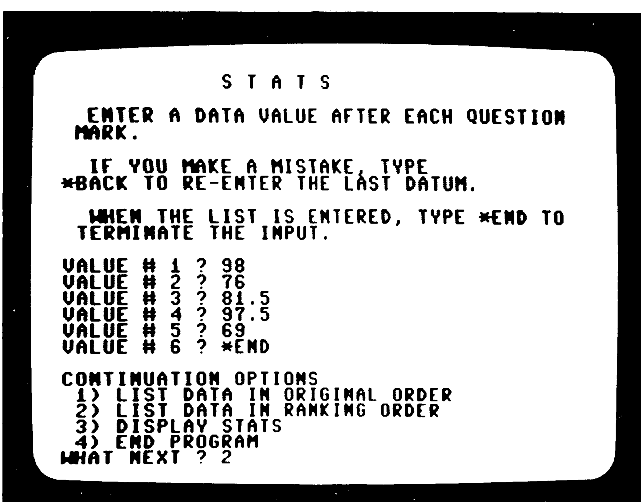
Here N is the number of values, V_i is value i, M is the mean value. The standard deviation is simply the square root of the variance. We do not have space to detail a lengthy discussion of their theoretical use. For this refer to the bibliography. Basically, however, the smaller the standard deviation, the more all the data tends to be clustered close to the mean value.

One word of warning—the first time option 2 or 3 is selected, the program must take some time to sort the data into numerical order. The time this requires depends upon how many items are on the list and how badly they are out of sequence. Average times are fifteen seconds for twenty-five items, about one minute for fifty items, about four minutes for a hundred items. The Commodore 64 will pause while this is occurring, so don't think it has hung up or fallen asleep! If you have several items on your list, this is the perfect chance to rob your refrigerator, make a quick phone call, or whatever.

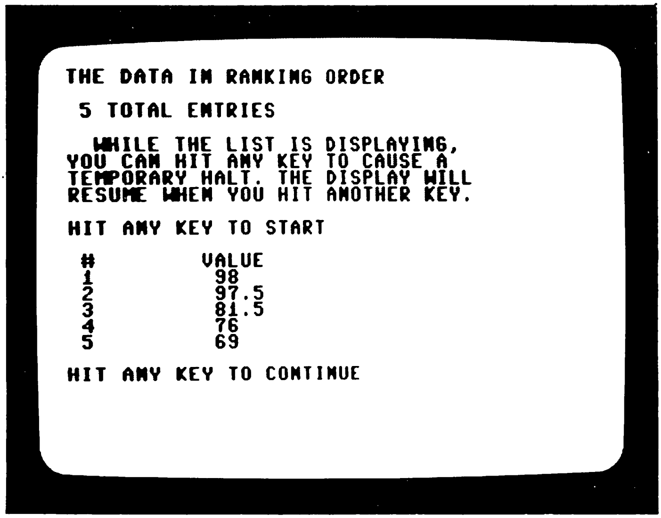
SAMPLE RUN



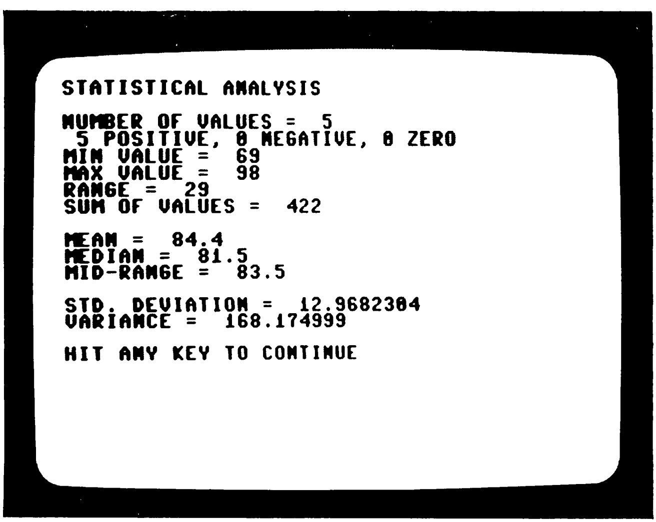
The program prompts the operator to begin entering the input data values.



The operator completes entering the scores of those who took a programming aptitude test. The actual test was given to many people, but for demonstration purposes, only five scores are used here. The special string, *END, is used to signal the end of the data. The operator then requests that the list be sorted into numerical order.



The operator hits a key to start the display and is shown the data list in ranking order. The program waits for a key to be pressed to continue.



Later in the run, the operator selects continuation option 3. This calculates and displays the various statistical quantities.

PROGRAM LISTING

```
READY.

100 REM: STATS
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
140 CLR
150 B$ = "*BACK":E$ = "*END"
160 MX = 100
170 DIM V(MX),Z(MX)
180 Z(0) = 0
200 PRINT CHR$(147):PRINT TAB(12);"S T A T S"
210 PRINT:PRINT"  ENTER A DATA";
220 PRINT" VALUE AFTER EACH";
230 PRINT" QUESTION MARK."
240 PRINT:PRINT"  IF YOU MAKE";
250 PRINT" A MISTAKE, TYPE"
260 PRINT B$;" TO RE-ENTER";
270 PRINT" THE LAST";
280 PRINT" DATUM.":PRINT
290 PRINT"  WHEN THE LIST IS";
300 PRINT" ENTERED, TYPE ";
310 PRINT E$;" TO TERMINATE";
320 PRINT" THE INPUT.":PRINT
500 N = 1
510 IF N < 1 THEN N = 1
520 PRINT"VALUE #";N;
530 INPUT R$:IF R$ = E$ THEN 700
540 IF R$ = B$ THEN N = N-1:GOTO 510
550 V(N) = VAL(R$)
560 IF N < MX THEN 600
570 PRINT"* NO MORE DATA";
580 PRINT" ALLOWED ! *": N= N+1
590 GOTO 700
600 N = N+1:GOTO 510
700 N = N-1
710 IF N > 0 THEN 800
720 PRINT"* NO DATA - RUN";
730 PRINT" ABORTED ! *"
740 END
800 PRINT
810 PRINT"CONTINUATION OPTIONS"
820 PRINT" 1) LIST DATA IN";
830 PRINT" ORIGINAL ORDER"
840 PRINT" 2) LIST DATA IN";
850 PRINT" RANKING ORDER"
860 PRINT" 3) DISPLAY STATS"
870 PRINT" 4) END PROGRAM"
880 INPUT"WHAT NEXT ";R$
```

```
890 R = INT(VAL(R$))
900 IF R < 1 OR R > 4 THEN 800
910 IF R = 4 THEN END
920 ON R GOSUB 1000,1200,1500
930 GOTO 800
1000 PRINT CHR$(147):PRINT"THE ORIGINAL";
1010 PRINT" DATA ORDER":PRINT
1020 PRINT N;"TOTAL ENTRIES"
1030 PRINT:GOSUB 2000:PRINT
1040 PRINT" #","VALUE"
1050 FOR J = 1 TO N:FOR K = 1 TO 100
1060 NEXT:PRINT J,V(J)
1070 GOSUB 2500:NEXT:GOSUB 2900
1080 RETURN
1200 PRINT CHR$(147):PRINT"THE DATA IN";
1210 PRINT" RANKING ORDER":PRINT
1220 PRINT N;"TOTAL ENTRIES"
1230 PRINT:GOSUB 2700:GOSUB 2000
1240 PRINT:PRINT" #","VALUE"
1250 FOR J = 1 TO N:FOR K = 1 TO 100
1260 NEXT:PRINT J,V(Z(J))
1270 GOSUB 2500:NEXT:GOSUB 2900
1280 RETURN
1500 PRINT CHR$(147):NP = 0:NN = 0:NZ = 0
1505 PRINT CHR$(147):SQ = 0:W = 0
1510 PRINT"STATISTICAL ANALYSIS"
1520 PRINT
1530 PRINT"NUMBER OF VALUES = ";N
1540 FOR J = 1 TO N:W = W+V(J)
1550 SQ = SQ+V(J)*V(J)
1560 IF V(J) > 0 THEN NP = NP+1
1570 IF V(J) < 0 THEN NN = NN+1
1580 IF V(J) = 0 THEN NZ = NZ+1
1590 NEXT:M = W/N:VA = 0
1600 IF N = 1 THEN 1620
1610 VA = (SQ-N*M*M)/(N-1)
1620 SD = VA/3:IF VA = 0 THEN 1650
1630 FOR J = 1 TO 65
1640 SD = (SD+VA/SD)/2:NEXT
1650 PRINT NP;"POSITIVE,";
1660 PRINT NN;"NEGATIVE,";
1670 PRINT NZ;"ZERO"
1680 GOSUB 2700
1690 PRINT"MIN VALUE = ";V(Z(N))
1700 PRINT"MAX VALUE = ";V(Z(1))
1710 Q = V(Z(1))-V(Z(N))
1720 PRINT"RANGE = ";Q
1730 PRINT"SUM OF VALUES = ";W
1740 PRINT:PRINT"MEAN = ";M
1750 Q = INT(N/2)+1:MD = V(Z(Q))
1760 IF N/2 > INT(N/2) THEN 1780
```

```
1770 MD = (V(Z(Q))+V(Z(Q-1)))/2
1780 PRINT"MEDIAN = ";MD
1790 Q = (V(Z(1))+V(Z(N)))/2
1800 PRINT"MID-RANGE = ";Q:PRINT
1810 PRINT"STD. DEVIATION = ";SD
1820 PRINT"VARIANCE = ";VA
1830 GOSUB 2900:RETURN
2000 PRINT"  WHILE THE LIST IS";
2010 PRINT" DISPLAYING,"
2020 PRINT"YOU CAN HIT ANY KEY";
2030 PRINT" TO CAUSE A"
2040 PRINT"TEMPORARY HALT. THE";
2050 PRINT" DISPLAY WILL"
2060 PRINT"RESUME WHEN YOU HIT";
2070 PRINT" ANOTHER KEY.":PRINT
2080 PRINT"HIT ANY KEY TO START"
2090 GET R$
2100 IF R$ = "" THEN 2090
2110 RETURN
2500 GET R$
2510 IF R$ = "" THEN RETURN
2520 GET R$
2530 IF R$ = "" THEN 2520
2540 RETURN
2700 IF Z(0) = 1 THEN RETURN
2710 FOR J = 1 TO N:Z(J) = J:NEXT
2720 IF N = 1 THEN RETURN
2730 NM = N-1:FOR K = 1 TO N
2740 FOR J = 1 TO NM:N1 = Z(J)
2750 N2 = Z(J+1)
2760 IF V(N1) > V(N2) THEN 2780
2770 Z(J+1) = N1:Z(J) = N2
2780 NEXT:NEXT:Z(0) = 1:RETURN
2900 PRINT
2910 PRINT"HIT ANY KEY";
2920 PRINT" TO CONTINUE";
2930 GET R$
2940 IF R$ = "" THEN 2930
2950 PRINT:RETURN

READY.
```

EASY CHANGES

1. The program arrays are currently dimensioned to allow a maximum of 100 data items. To achieve up to 1000 data items, make this change:

   160 MX = 1000

2. You may wish to change the special strings that signal termination of data input and/or the backing up of data input. These are controlled by the variables E$ and B$, respectively. They are set in line 150. If you wish to terminate the data with /DONE/ and to back up with /LAST/ for example, line 150 should be:

   ```
   150 B$="/LAST/":E$="/DONE/"
   ```

3. You may wish to see your lists sorted from smallest value to largest value instead of the other way around, as done now. This can be accomplished by changing the "greater than" sign (>) in line 2760 to a "less than" sign (<). Thus:

   ```
   2760 IF V(N1)<V(N2) THEN 2780
   ```

   This will, however, cause a few funny things to happen to the statistics. The real minimum value will be displayed under the heading "maximum" and vice-versa. Also, the range will have its correct magnitude but with an erroneous minus sign in front. To cure these afflictions, make these changes also:

   ```
   1690 PRINT "MIN VALUE=";V(Z(1))
   1700 PRINT "MAX VALUE=";V(Z(N))
   1710 Q=V(Z(N))-V(Z(1))
   ```

MAIN ROUTINES

140- 180	Initializes constants and dimensioning.
200- 320	Displays messages.
500- 600	Gets data from the user.
700- 740	Checks that input contains at least one value.
800- 930	Command mode—gets user's next option and does a GOSUB to it.
1000-1080	Subroutine to list data in the original order.
1200-1280	Subroutine to list data in ranking order.
1500-1830	Subroutine to calculate and display statistics.
2000-2110	Subroutine to display various messages.
2500-2540	Subroutine to allow user to temporarily start and stop display listing.
2700-2780	Subroutine to sort the list in ranking order.
2900-2950	Subroutine to detect if user has hit a key to continue.

MAIN VARIABLES

MX	Maximum number of data values allowed.
V(MX)	Array of the data values.
Z(MX)	Array of the sorting order.
N	Number of data values in current application.
B$	Flag string to back up the input.
E$	Flag string to signal end of the input.
R$	User input string.
NM	N − 1.
R	Continuation option.
NP	Number of positive values.
NN	Number of negative values.
NZ	Number of zero values.
W	Sum of the values.
SQ	Sum of the squares of the values.
M	Mean value.
MD	Median of the values.
VA	Variance.
SD	Standard deviation.
J,K	Loop indices.
N1,N2	Possible data locations to interchange during sorting.
Q	Work variable.

SUGGESTED PROJECTS

1. The sorting algorithm used in the program is efficient only when the number of list items is fairly small—less than twenty-five or so. This is because it does not do checking along the way to see when the list becomes fully sorted. If your lists tend to be longer than twenty-five items, you might wish to use another sorting algorithm more appropriate for longer lists. Try researching other sorts and incorporating them into the program. To get you started, try these changes:

```
2730 Q=0: FOR J=1 TO N-1:N1=Z(J)
2740 N2=Z(J+1)
2750 IF V(N1)> =V(N2) THEN 2780
2760 Z(J+1)=N1:Z(J)=N2
2770 Q=1
2780 NEXT:IF Q=1 THEN 2730
2790 Z(0)=1:RETURN
```

If your lists are short, this routine will probably be a little slower than the current one. However, for longer lists it will save proportionately more and more time.

2. Many other statistical parameters exist to describe this kind of data. Research them and add some that might be useful to you. One such idea is classifying the data. This consists of dividing the range into a number of equal classes and then counting how many values fall into each class.

Section 6

Miscellaneous Programs

INTRODUCTION TO MISCELLANEOUS PROGRAMS

These programs show how simple programs can do interesting things. Most of them have a mathematical flavor. They are short and, as such, would be useful for study for those just learning BASIC in particular or programming in general.

Monte Carlo simulation involves programming the computer to conduct an experiment. (It doesn't involve high-stakes gambling!) PI shows how this technique can be used to calculate an approximation to the famous mathematical constant pi.

PYTHAG will find all right triangles with integral side lengths. A clever algorithm is utilized to do this.

Have you ever looked around your classroom or club meeting and wondered if any two people had the same birthdate? BIRTHDAY will show you what the surprising odds are.

Very high precision arithmetic can be done on the Commodore 64 with the proper "know-how." POWERS will calculate the values of integers raised to various powers; not to the computer's standard nine digit precision, but up to 250 full digits of precision.

Your computer can play music! TUNE allows you to enter tunes into your computer in a simple, convenient manner. Then your computer will play them for you.

BIRTHDAY

PURPOSE

Suppose you are in a room full of people. What is the probability that two or more of these people have the same birthday? How many people have to be in the room before the probability becomes greater than 50 percent? We are talking only about the month and day of birth, not the year.

This is a fairly simple problem to solve, even without a computer. With a computer to help with the calculations, it becomes very easy. What makes the problem interesting is that the correct answer is nowhere near what most people immediately guess. Before reading further, what do you think? How many people have to be in the room before there is better than a 50-50 chance of birthday duplication? 50? 100? 200?

HOW TO USE IT

When you RUN the program, it starts by displaying headings over two columns of numbers that will be shown. The left column is the number of people in the room, starting with one. The right column is the probability of birthday duplication.

For one person, of course, the probability is zero, since there is no one else with a possible duplicate birthday. For two people, the probability is simply the decimal equivalent of $1/365$ (note that we assume a 365 day year, and an equal likelihood that each person could have been born on any day of the year).

What is the probability of duplication when there are three people in the room? No, not just $2/365$. It's actually

$$1 - (364/365 \text{ times } 363/365)$$

This is simply one minus the probability of *no* duplicate birthdays.

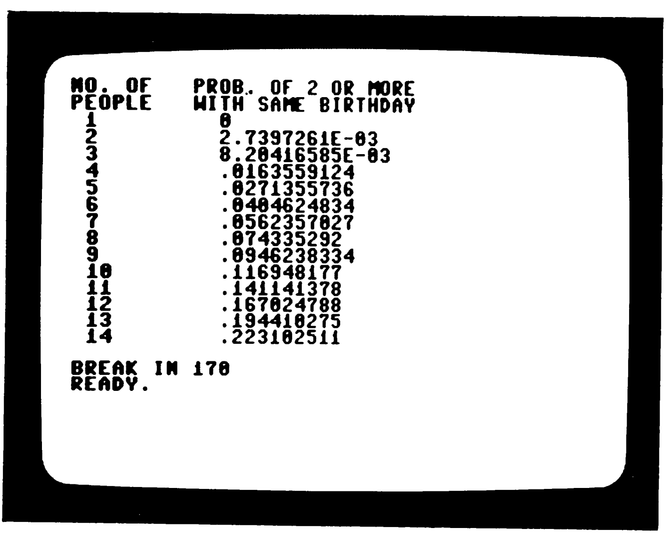
The probability for four people is

$$1 - (^{364}/_{365} \text{ times } ^{363}/_{365} \text{ times } ^{362}/_{365})$$

The calculation continues like this, adding a new term for each additional person in the room. You will find that the result (probability of duplication) exceeds .50 surprisingly fast.

The program continues with the calculation until there are 60 people in the room. You will have to **RUN/STOP** the program long before that to see the point where the probability first exceeds 50 percent.

SAMPLE RUN



After the probability of 14 people with the same birthday is shown, the RUN/STOP key is pressed to terminate the run.

PROGRAM LISTING

```
READY.

100 REM: BIRTHDAY
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 PRINT CHR$(147)
130 PRINT"NO. OF   PROB. OF 2 OR MORE"
140 PRINT"PEOPLE   WITH SAME BIRTHDAY"
150 Q=1
160 FOR N=1 TO 60
170 PRINT N,1-Q
180 Q=Q*(365-N)/365
190 NEXT N
200 END

READY.
```

EASY CHANGES

Change the constant value of 60 at the end of line 160 to alter the range of the number of people in the calculation. For example, change it to 100 and watch how fast the probability approaches 1.

MAIN ROUTINES

120-140	Displays headings.
150	Initializes Q to 1.
160-190	Calculates probability of no duplication, then displays probability of duplication.

MAIN VARIABLES

N	Number of people in the room.
Q	Probability of no duplication of birthdays.

SUGGESTED PROJECTS

Modify the program to allow for leap years in the calculation, instead of assuming 365 days per year.

PI

PURPOSE AND DISCUSSION

The Greek letter pi, π, represents probably the most famous constant in mathematical history. It occurs regularly in many different areas of mathematics. It is best known as the constant appearing in several geometric relationships involving the circle. The circumference of a circle of radius r is $2\pi r$, while the area enclosed by the circle is πr^2.

Being a transcendental number, pi cannot be expressed exactly by any number of decimal digits. To nine significant digits, its value is 3.14159265. Over many centuries, man has devised many different methods to calculate pi.

This program uses a valuable, modern technique known as computer simulation. The name "simulation" is rather self-explanatory; the computer performs an experiment for us. This is often desirable for many different reasons. The experiment may be cheaper, less dangerous, or more accurate to run on a computer. It may even be impossible to do in "real life." Usually, however, the reason is that the speed of the computer allows the simulation to be performed many times faster than actually conducting the real experiment.

This program simulates the results of throwing darts at a specially constructed dartboard. Consider Figure 1 which shows the peculiar square dartboard involved. The curved arc, outlining the shaded area, is that of a circle with the center in the lower left hand corner. The sides of the square, and thus the radius of the circle, are considered to have a length of 1.

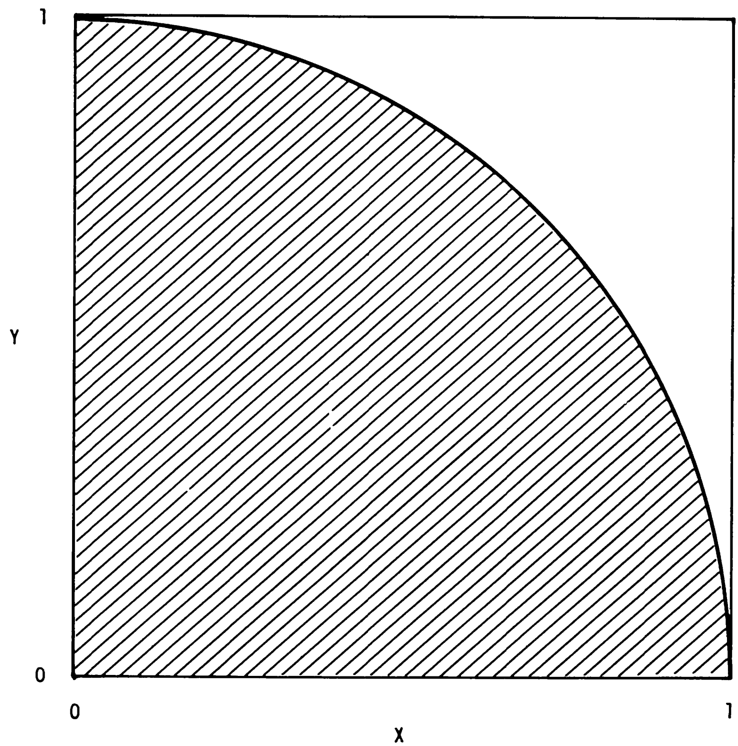


Figure 1. The Pi Dartboard.

Suppose we were able to throw darts at this square target in such a way that each dart had an equal chance of landing anywhere within the square. A certain percentage of darts would result in "hits," i.e. land in the shaded area. The expected value of this percentage is simply the area of the shaded part divided by the area of the entire square.

The area of the shaded part is one fourth of the area the entire circle would enclose if the arc were continued to completely form the circle. Recall the area of a circle is πr^2 where r is the radius. In our case, $r = 1$, and the area of the entire circle would simply be π. The shaded area of the dartboard is one fourth of this entire circle and thus has an area of $\pi/4$. The area of the square is s^2, where s is the length of a side. On our dartboard, $s = 1$, and the area of the whole dartboard is 1.

Now the expected ratio of "hits" to darts thrown can be expressed

$$\text{RATIO} = \frac{\#\text{ hits}}{\#\text{ thrown}} = \frac{\text{shaded area}}{\text{entire area}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

So we now have an experimental way to approximate the value of π. We perform the experiment and compute the ratio of "hits" observed. We then multiply this number by 4 and we have calculated π experimentally.

But instead of actually constructing the required dartboard and throwing real darts, we will let the Commodore 64 do the job. The program "throws" each dart by selecting a separate random number between 0 and 1 for the X and Y coordinates of each dart. This is accomplished by using the built-in RND function of Basic. A "dart" is in the shaded area if $X^2 + Y^2 < 1$ for it.

So the program grinds away, continually throwing darts and determining the ratio of "hits." This ratio is multiplied by 4 to arrive at an empirical approximation to π.

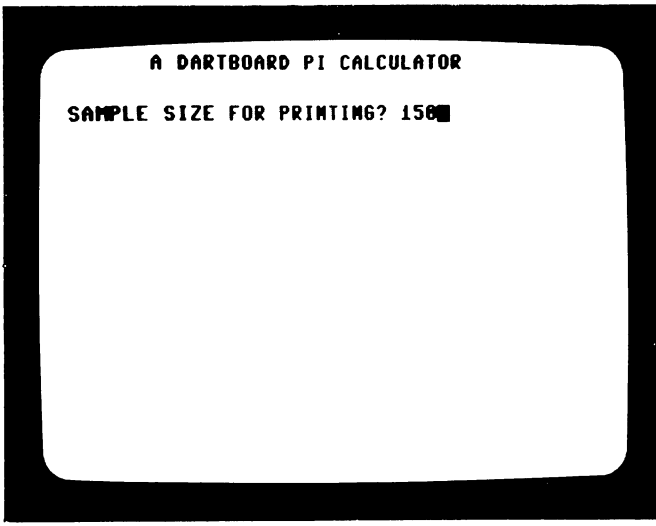
HOW TO USE IT

The program requires only one input from you. This is the "sample size for printing," i.e. how many darts it should throw before printing its current results. Any value of one or higher is acceptable.

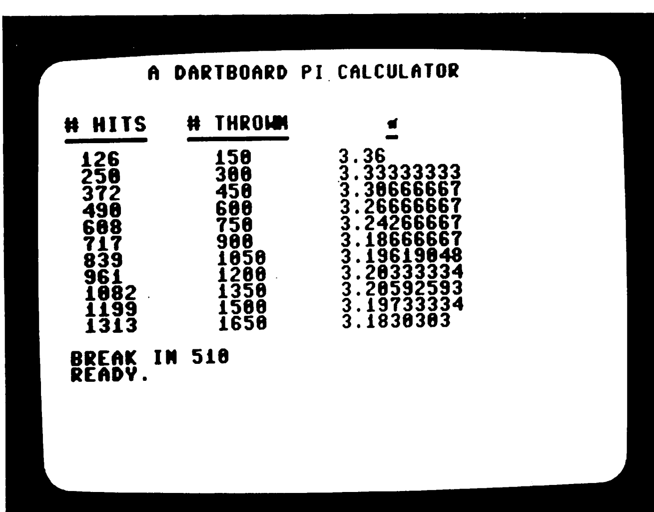
After you input this number, the program will commence the simulation and display its results. A cumulative total of "hits," darts thrown, and the current approximation to π will be displayed for each multiple of the sample size.

This will continue until you press the **RUN/STOP** key. When you are satisfied with the total number of darts thrown, press the **RUN/STOP** key to terminate the program execution.

SAMPLE RUN



The operator selects 150 for the printing sample size.



After 1650 darts are "thrown", the user presses the RUN/STOP key to terminate the run.

PROGRAM LISTING

```
READY.

100 REM: PI
110 REM: COPYRIGHT 1983
111 REM: PHIL FELDMAN, TOM RUGG, AND
112 REM: WESTERN SYSTEMS GROUP
150 Q=RND(-TI)
160 T=0:TH=0
300 GOSUB 600
310 INPUT"SAMPLE SIZE FOR PRINTING";NP
320 NP=INT(NP):IF NP<1 THEN 300
330 GOSUB 600
340 PRINT"# HITS   # THROWN";TAB(25);CHR$(255)
360 FOR J=1 TO 6:GOSUB 700:NEXT
370 PRINT TAB(9);
375 FOR J=1 TO 8:GOSUB 700:NEXT:PRINT TAB(25);
380 GOSUB 700:PRINT
400 GOSUB 500:TH=TH+NH:T=T+NP:P=4*TH/T
410 PRINT TH,T,P
420 GOTO 400
500 NH=0:FOR J=1 TO NP
510 X=RND(1):Y=RND(1)
520 IF (X*X+Y*Y)<1 THEN NH=NH+1
530 NEXT:RETURN
600 PRINT CHR$(147);TAB(6);
610 PRINT"A DARTBOARD PI CALCULATOR"
620 PRINT:PRINT:RETURN
700 PRINT CHR$(196);:RETURN

READY.
```

EASY CHANGES

1. If you want the program to always use a fixed sample size, change line 310 to read

   310 NP=150:GOTO 370

   Of course, the value of 150 given here may be changed to whatever you wish.
2. If you want the program to stop by itself after a certain number of darts have been thrown, add the following two lines:

   315 INPUT"TOTAL # DARTS TO THROW";ND
   415 IF T>=ND THEN END

This will ask the operator how many total darts should be thrown, and then terminate the program when they have been thrown.

MAIN ROUTINES

150-160	Initializes constants.
300-380	Gets operator input, displays column headings.
400-420	Calculates and displays results.
500-530	Throws NP darts and records number of "hits."
600-620	Clears screen and displays program title.
700	Draws underlining characters.

MAIN VARIABLES

T	Total darts thrown.
TH	Total "hits."
NP	Sample size for printing.
NH	Number of hits in one group of NP darts.
P	Calculated value of pi.
Q	Work variable.
X,Y	Random-valued coordinates of a dart.
J	Loop index.

SUGGESTED PROJECTS

1. Calculate the percentage error in the program's calculation of pi and display it with the other results. The percentage error, PE, can be calculated as:

$$PE = 100*ABS(P - \pi)/\pi$$

2. The accuracy of this simulation is highly dependent on the quality of the computer's random number generator. Try researching different algorithms for pseudo random number generation. Then try incorporating them into the program. Change line 510 to use the new algorithm(s). This can actually be used as a test of the various random number generators. Gruenberger's book, referenced in the bibliography, contains good material on various pseudo random number generators.

POWERS

PURPOSE

By now you have probably learned that the Commodore 64 keeps track of nine significant digits when dealing with numbers. For integers less than one billion (1,000,000,000), the Commodore 64 can retain the precise value of the number. But for larger integers the Commodore 64 only keeps track of the most significant (leftmost) nine digits, plus the exponent. This means, of course, that there is no way you can use the computer to deal with precise integers greater than one billion, right?

Wrong.

This program calculates either factorials or successive powers of an integer, and can display precise results that are up to 250 digits long. By using a "multiple-precision arithmetic" technique, this program can tell you *exactly* what 973 to the 47th power is, for example.

HOW TO USE IT

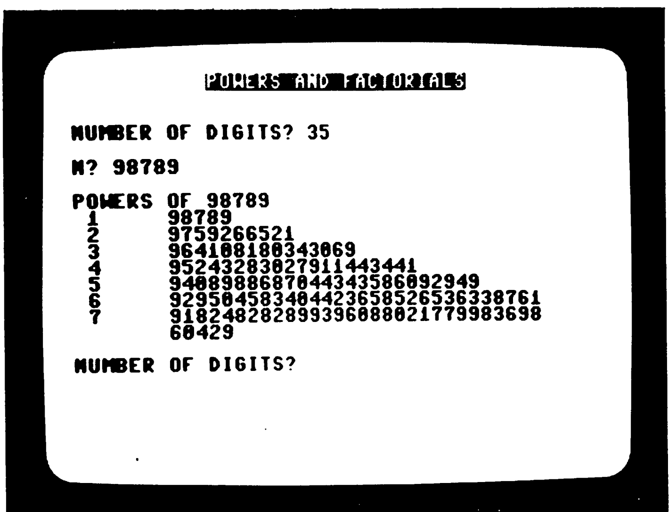
The program first asks you how many digits long you want the largest number to be. This can be any integer from 1 to 250. So, for example, if you enter 40, you will get answers up to forty digits long.

Next you are asked for the value of N. If you respond with a value of 1, you are requesting to be shown all the factorials that will fit in the number of digits you specified. First you will get one factorial, then two factorial, and so on. In case you have forgotten, three factorial is 3 times 2 times 1, or 6. Four factorial is 4 times 3 times 2 times 1, or 24.

If you enter an N in the range from 2 through 100,000, you are requesting the successive powers of that number up to the limit of digits you specified. So, if you provide an N of 23, you will get 23 to the first power, then 23 squared, then 23 cubed, and so on.

Finally, after it has displayed the largest number that will fit within the number of digits you entered, the program starts over. The larger the number of digits you ask for, the longer it will take the program to calculate each number. If you enter zero, the program ends.

SAMPLE RUN



The operator wants answers up to 35 digits long in the calculations of the powers of 98789. The program calculates numbers up to 98789^7 and then asks for the number of digits again (in preparation for the next calculation the operator requests).

PROGRAM LISTING

```
READY.

100 REM: POWERS
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 PRINT CHR$(147)
130 PRINT TAB(10);CHR$(18);
140 PRINT"POWERS AND FACTORIALS"
150 PRINT:PRINT
160 DIM N(255)
170 INPUT"NUMBER OF DIGITS";M
180 M=INT(M):IF M>250 THEN 170
185 IF M < 1 THEN END
190 PRINT:INPUT"N";N
200 N=INT(N)
210 IF N>100000 THEN 190
215 IF N < 1 THEN 190
220 PRINT
230 F=0:IF N=1 THEN F=1:PRINT"FACTORIALS"
240 IF F=0 THEN PRINT"POWERS OF";N
250 T=10:K=1:N(0)=N
260 FOR J=0 TO M
270 IF N(J)<T THEN 300
280 Q=INT(N(J)/T):W=N(J)-Q*T
290 N(J)=W:N(J+1)=N(J+1)+Q
300 NEXT
310 J=M+1
320 IF N(J)=0 THEN J=J-1:GOTO 320
330 IF J>=M THEN 500
340 D=0:PRINT K;TAB(7);
350 N$=STR$(N(J)):N$=RIGHT$(N$,1)
360 D=D+1:IF D>30 THEN D=1:PRINT:PRINT TAB(7);
370 PRINT N$;:J=J-1:IF J>=0 THEN 350
380 IF F=1 THEN N=N+1
390 K=K+1:PRINT
400 FOR J=0 TO M:N(J)=N(J)*N:NEXT
410 GOTO 260
500 FOR J=1 TO 255:N(J)=0:NEXT
510 PRINT:GOTO 170

READY.
```

EASY CHANGES

1. To change the program so that it always uses, say, fifty digit numbers, remove lines 170 and 180, and insert this line:

    170 M=50

2. To clear the screen before the output begins being displayed, insert this line:

```
220 PRINT CHR$(147)
```

MAIN ROUTINES

120-160	Displays title. Sets up array for calculations.
170-240	Asks for number of digits and N. Checks validity of responses. Displays heading.
250	Initializes variables for calculations.
260-300	Performs "carrying" in N array so each element has a value no larger than 9.
310-320	Scans backwards through N array for first non-zero element.
330	Checks to see if this value would be larger than the number of digits requested.
340-370	Displays counter and number. Goes to second line if necessary.
380-390	Prepares to multiply by N to get next number.
400-410	Multiplies each digit in N array by N. Goes back to line 260.
500-510	Zeroes out N array in preparation for next request. Goes back to 170.

MAIN VARIABLES

N	Array in which calculations are made.
M	Number of digits of precision requested by operator.
N	Starting value. If 1, factorials. If greater than 1, powers of N.
F	Set to zero if powers, 1 if factorials.
T	Constant value of 10.
K	Counter of current power or factorial.
J	Subscript variable.
Q,W	Temporary variables used in reducing each integer position in the N array to a value from 0 to 9.
D	Number of digits displayed so far on the current line.
N$	String variable used to convert each digit into displayable format.

SUGGESTED PROJECTS

1. Determine the largest N that could be used without errors entering into the calculation (because of intermediate results exceeding one billion), then modify line 210 to permit values that large to be entered.
2. Create a series of subroutines that can add, subtract, multiply, divide, and exchange numbers in two arrays, using a technique like the one used here. Then you can perform high precision calculations by means of a series of GOSUB statements.

PYTHAG

PURPOSE

Remember the Pythagorean Theorem? It says that the sum of the squares of the two legs of a right triangle is equal to the square of the hypotenuse. Expressed as a formula, it is $a^2 + b^2 = c^2$. The most commonly remembered example of this is the 3-4-5 right triangle ($3^2 + 4^2 = 5^2$). Of course, there are an infinite number of other right triangles.

This program displays integer values of a, b, and c that result in right triangles.

HOW TO USE IT

To use this program, all you need to do is RUN it and watch the "Pythagorean triplets" (sets of values for a, b, and c) come out. The program displays twenty sets of values on each screen, and then waits for you to press any key (except **RUN/STOP**) before it continues with the next twenty. It will go on indefinitely until you press the **RUN/STOP** key.

The left-hand column shows the count of the number of sets of triplets produced, and the other three columns are the values of a, b, and c.

The sequence in which the triplets are produced is not too obvious, so we will explain how the numbers are generated.

It has been proved that the following technique will generate all *primitive* Pythagorean triplets. ("Primitive" means that no set is an exact multiple of another.) If you have two positive integers called R and S such that:

1. R is greater than S,
2. R and S are of opposite parity (one is odd and the other is even), and
3. R and S are relatively prime (they have no common integer divisors except 1),

then a, b, and c can be found as follows:

$$a = R^2 - S^2$$
$$b = 2RS$$
$$c = R^2 + S^2$$

The program starts with a value of 2 for R. It generates all possible S values for that R (starting at $R - 1$ and then decreasing) and then adds one to R and continues. So, the first set of triplets is created when R is 2 and S is 1, the second set when R is 3 and S is 2, and so on.

SAMPLE RUN

```
**** PYTHAGOREAN TRIPLETS ****

COUNT      --A--      --B--      --C--
1          3          4          5
2          5          12         13
3          7          24         25
4          15         8          17
5          9          40         41
6          21         20         29
7          11         60         61
8          35         12         37
9          13         84         85
10         33         56         65
11         45         28         53
12         15         112        113
13         39         80         89
14         55         48         73
15         63         16         65
16         17         144        145
17         65         72         97
18         77         36         85
19         19         180        181
20         51         140        149

PRESS ANY KEY TO CONTINUE
```

The program shows the first screen of Pythagorean triplets.

PROGRAM LISTING

```
READY.

100 REM: PYTHAG
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
130 R=2:K=1:D=0
150 GOSUB 350
180 S=R-1
190 A=R*R-S*S
200 B=2*R*S
210 C=R*R+S*S
220 PRINT K,A,B,C
230 K=K+1:D=D+1:GOTO 400
240 S=S-2:IF S<=0 THEN R=R+1:GOTO 180
250 S1=S
255 B1=R
260 N=INT(B1/S1)
270 R1=B1-(S1*N)
280 IF R1<>0 THEN B1=S1:S1=R1:GOTO 260
300 IF S1<>1 THEN 240
320 GOTO 190
350 PRINT CHR$(147);
360 PRINT"**** PYTHAGOREAN TRIPLETS ****"
370 PRINT
380 PRINT"COUNT","---A---","---B---","---C---"
390 RETURN
400 IF D<20 THEN 240
410 PRINT
420 PRINT"PRESS ANY KEY TO CONTINUE";
430 GET R$:IF R$="" THEN 430
440 GOSUB 350
450 D=0
460 GOTO 240

READY.
```

EASY CHANGES

1. Alter the starting value of R in line 130. Instead of 2, try 50 or 100.

2. If you want, you can change the number of sets of triplets displayed on each screen. Change the 20 in line 400 to a 10, for example. You probably won't want to try a value greater than 20, since that would cause the column headings to roll off the screen.
3. To make the program continue without requiring you to press a key for the next screen of values, insert either of these lines:

```
405 GOTO 440
```

or

```
405 GOTO 450
```

The first will display headings for each screen. The second will only display the headings at the beginning of the run.

MAIN ROUTINES

130	Initializes variables.
150	Displays the title and column headings.
180	Calculates first value of S for current R value.
190-210	Calculates A, B, and C.
220-230	Displays one line of values. Adds to counters.
240	Calculates next S value. If no more, calculates next R value.
250-300	Determines if R and S are relatively prime.
350-390	Subroutine to display title and column headings.
400-460	Checks if screen is full yet. If so, waits for key to be pressed.

MAIN VARIABLES

R,S	See explanation in "How To Use It."
K	Count of total number of sets displayed.
D	Count of number of sets displayed on one screen.
A,B,C	Lengths of the three sides of the triangle.
S1,B1, R1,N	Used in determining if R and S are relatively prime.
R$	Key pressed by operator to continue.

SUGGESTED PROJECTS

1. In addition to displaying K, A, B, and C on each line, display R and S. You will have to squeeze the columns closer together.
2. Because this program uses integer values that get increasingly large, eventually some will exceed the Commodore 64's integer capacity and produce incorrect results. Can you determine when this will be? Modify the program to stop when this occurs.

TUNE

PURPOSE

The Commodore 64 has tremendous sound capabilities. This program allows you to experiment with simple computer tune-playing.

HOW TO USE IT

As shown in the Program Listing, TUNE currently plays a familiar portion of Puff, The Magic Dragon. We'll explain how to enter other tunes in a moment.

When you RUN the program, it displays the title of the program at the top of the screen and immediately begins playing its tune. Of course, you have to be using a television with a speaker, and its volume control needs to be set loud enough for you to hear the music.

As each note is played, a character is plotted on the TV screen, starting from the left and moving to the right. The vertical position of the point corresponds with the pitch of the note—high notes are near the top, and low notes near the bottom. If the tune has more than 20 notes, the display goes to the right edge of the screen and begins overlaying the notes at the left edge. The characters displayed are: W = whole note, D = dotted half note, H = half note, Q = quarter note, E = eighth note, and R = rest.

This graphics display gives you something to watch while the tune is being played, and for many tunes it clearly shows you the patterns and symmetry that make music enjoyable.

After you get tired of hearing Puff, you will undoubtedly want to enter some tunes of your own choosing. With only a lit-

tle musical training, you can translate a simple piece of sheet music into the notation needed by this program. We may be crazy, but we think we can give you enough of an introduction to reading music in the next few paragraphs that you will be able to figure out how to understand simple musical notation and copy a tune into this program. For more details on reading music, refer to an introductory music text.

First, turn to Appendix M of your *Commodore 64 User's Guide* manual. The main thing to learn from Appendix M is the frequencies for each musical note. These frequencies will be used as POKE values within the program.

The next thing we need to know is the duration of each note. A quarter note is represented by a solid black oval (note-head) with a "stick" (stem) attached. The most common notes in most music are whole notes, half notes, quarter notes and eighth notes. The duration of each of these notes, relative to the others, is just as you would expect from their fraction-like names. For example, a quarter note lasts twice as long as an eighth note, but only half as long as a half note.

A half note looks like a hollow quarter note (i.e., an oval on a stick). A whole note is an oval with no stick. An eighth note looks like a quarter note with a flag on the stick. If two eighth notes are next to each other, they are usually connected by one bar rather than having individual flags.

A *chord* is several notes played simultaneously. It looks like a stack of ovals on one stick. The Commodore 64 can play chords, but we have chosen to make that one of your special projects. For our example, we will play the top note on the stack when we encounter a chord. This is the simplest way to follow the melody like a one-finger piano player.

And now a word about sharps and flats. The white keys on a piano are the *natural* notes. The black keys are the *accidentals* (sharps and flats). On a musical score, sharps are indicated by the pound sign ("#"), and flats by a symbol that resembles a lower case "b." Naturals are assumed when neither the sharp or flat symbol is used. When a particular note is always sharp, for example, in a tune, this is indicated at the far left of the musical staff (the five parallel lines upon which the notes are drawn), rather than next to every occurrence of the note. This applies to any octave the note is played in. For example, if F sharp is indicated at the left of the staff, all F's are sharp, whether on that same line of the staff or not. When an F should be a natural in-

stead of a sharp, the natural symbol is used next to the F. The natural symbol is drawn in various ways, but generally looks like a little rectangle.

With these fundamentals in mind, let's look at how we enter music into the TUNE program. The DATA statements that represent the tune to be played are located between lines 1000 and 2999 of the program. To enter your own music, delete those lines so you can begin entering yours. Leave line 3000 as it is.

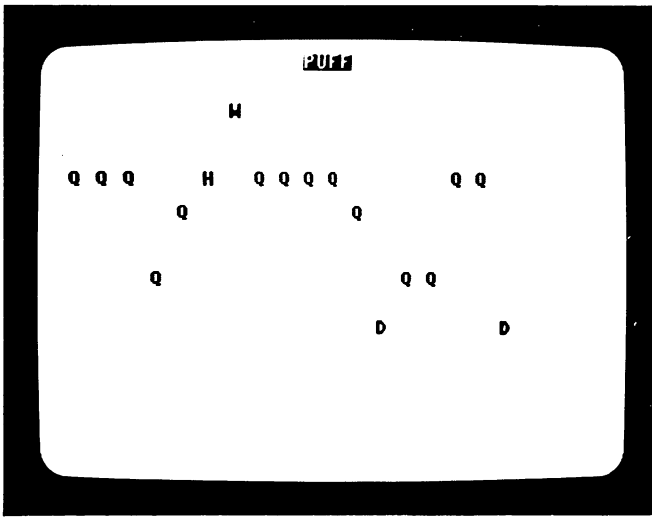
For each note of your tune, the computer needs to know two things: the note to be played and its duration. The DATA statements you enter after line 1000 provide this information. Each entry you make is separated by commas, and indicates high frequency, low frequency and duration. For each note to be played, 3 data items are supplied; high frequency (0 for rest), low frequency (0 for rest) and duration. Duration is specified as; 4 = whole note, 3 for a dotted half note, 2 for a half note, 1 for a quarter note, and .5 for an eighth note. The other items the Commodore 64 requires to make music are ATTACK/DECAY, SUSTAIN/RELEASE, WAVEFORM and VOLUME. VOLUME is set at the beginning of the program in line 200. ATTACK/DECAY is set at 136 (ATK4(128) and DEC4(8)). SUSTAIN/RELEASE is set at 136 (SUS4(128) and REL4(8)). The WAVEFORM is set at 17 (Triangle). These values are arbitrary on our part and you will probably want to experiment with them (See EASY CHANGES). The Commodore 64 User's Guide (Chapter 7) contains an extensive discussion on creating sound with the Commodore 64.

That covers all the rules. All you need to do is enter the DATA statements for your tune after line 1000. It is generally a good idea to leave some space between your line numbers, in case you discover later that you need to add some more lines. Starting with line 1000, and adding 10 or 20 for each new line number is a good approach.

Note that we used a separate DATA statement for each *bar* (or *measure*, the space between two vertical lines on the staff), which helps keep things organized in case we have to go back to fix a mistake.

There is a great deal more about music that we cannot cover in this limited space, but we hope this introduction has been enough to get you going. Be sure to read the Easy Changes section of this chapter to see several useful modifications you can easily make.

SAMPLE RUN



The program begins playing its tune and displaying a letter for each note sounded.

PROGRAM LISTING

```
READY.

100 REM: TUNE
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 N(1)=5:N(2)=17:N(3)=8:N(4)=4:N(5)=23
130 C(1)=3:C(2)=4:C(3)=5:C(4)=6:C(5)=7
140 POKE 53280,9:POKE 53281,2
180 X=42:S=1024:C=55296
190 DT = 300
200 POKE 54296,15
210 READ HI,LO,DU
220 IF HI < 999 THEN 230
225 POKE 54296,0:POKE 53280,14:POKE 53281,6
226 PRINT CHR$(147):END
230 DL = DU*DT
```

```
235 GOSUB 400:GOSUB 500
240 POKE 54277,136
250 POKE 54278,136
260 POKE 54273,HI
270 POKE 54272,LO
280 POKE 54276,17
290 FOR D = 1 TO DL:NEXT
300 POKE 54277,0
310 POKE 54278,0
320 POKE 54273,0
330 POKE 54272,0
340 POKE 54276,0
360 GOTO 210
400 NO = 18:CO = 0:IF HI = 0 THEN RETURN
410 IN = INT(DU+.5)+1
420 NO = N(IN)
430 CO = C(IN)
440 RETURN
500 Y = 41-HI:IF HI <= 0 THEN Y = 2
510 X = X + 2:IF X < 40 THEN 520
515 PRINT CHR$(147);TAB(18);CHR$(18);"PUFF":X=0
520 POKE S+X+Y*40,NO
530 POKE C+X+Y*40,CO
540 RETURN
1000 DATA 34,75,1,0,0,.5,34,75,.5,34,75,1,34,75,1
1010 DATA 32,94,2,25,177,1,0,0,1
1020 DATA 28,214,2,34,75,1,34,75,1
1030 DATA 25,177,3,25,177,1
1040 DATA 22,227,1,22,227,1,25,177,1,22,227,1
1050 DATA 21,154,1,25,177,1,34,75,1
1055 DATA 34,75,.5,34,75,.5
1060 DATA 34,75,1,28,214,1,32,94,.5,34,75,1.5
1070 DATA 38,126,4
1080 DATA 34,75,1,34,75,1,34,75,1,34,75,1
1090 DATA 32,94,1,25,177,3
1100 DATA 28,214,1,28,214,1,34,75,1,34,75,1
1110 DATA 25,177,3,25,177,1
1120 DATA 22,227,1,22,227,1,25,177,1,22,227,1
1130 DATA 21,154,1,25,177,1,34,75,1.5,34,75,.5
1140 DATA 28,214,.5,34,75,1.5,32,94,1,38,126,1
1150 DATA 34,75,2,0,0,1,32,94,1
1160 DATA 34,75,1,0,0,.5,34,75,.5,34,75,1,34,75,1
1170 DATA 32,94,2,25,177,1,0,0,1
1180 DATA 28,214,2,34,75,1,34,75,1
1190 DATA 25,177,3,25,177,1
1200 DATA 22,227,1,22,227,1,25,177,1,22,227,1
1210 DATA 21,154,1,25,177,1,34,75,1
1215 DATA 34,75,.5,34,75,.5
1220 DATA 34,75,1,28,214,1,32,94,.5,34,75,1.5
1225 DATA 38,126,4
1230 DATA 34,75,1,0,0,.5,34,75,.5,34,75,1,34,75,1
```

```
1240 DATA 32,94,2,25,177,1,0,0,1
1250 DATA 28,214,2,34,75,1,34,75,1
1260 DATA 25,177,3,25,177,1
1270 DATA 22,227,1,22,227,1,25,177,1,22,227,1
1280 DATA 21,154,1,25,177,1,34,75,.5
1285 DATA 34,75,.5,34,75,.5
1290 DATA 28,214,2,34,75,2
1300 DATA 32,94,2,38,126,2
1310 DATA 34,75,4
3000 DATA 999,999,4

READY.
```

EASY CHANGES

1. To change the *tempo* (speed) of the tune, change the value of DT in line 190. Larger values cause all the notes to be held longer, slowing down the tune. For example, to make the tune go faster, change line 190 to look like this:

   190 DT=200

   To make it go slower, use:

   190 DT=400
2. Lines 240, 250 and 280 set the ATTACK/DECAY, SUSTAIN/RELEASE and WAVEFORM settings, respectively. Using Appendix P of the Commodore 64 User's Guide, experiment with these settings to see if a more pleasing sound can be obtained.

MAIN ROUTINES

120- 200	Initializes variables, clears screen, sets background color.
210- 220	Reads DATA statements. Determines if at end of tune.
230	Calculates duration of note.
235	Performs subroutines to set graphics character and color of note to be played.
240- 290	Plays note and holds for calculated duration.
300- 360	Stops playing note and returns to get next note.
400- 440	Subroutine to get graphic for note to be played and color to be used for graphic.
500- 540	Subroutine to determine graphic position on screen. Also tests for screen overflow and resets with title.
1000-3000	DATA statements for tune being played.

MAIN VARIABLES

N	Array of graphics characters.
C	Array of color codes associated with each graphics character.
X	Current screen position (X-axis).
Y	Current screen position (Y-axis).
NO	Current graphics character.
CO	Current graphics color.
DT	Base number for calculation of duration of each note.
DL	Duration of current note.
HI	High frequency of current note.
LO	Low frequency of current note.
DU	Basic duration of current note.
D	Loop variable.
S	Start of CRT area for graphics.
C	Start of CRT area for color.

SUGGESTED PROJECTS

1. Write your own tune, using the format supplied. Add support for sixteenth notes to the program and graphics display.
2. As shown, this program only uses VOICE 1. Modify the program to use VOICE 2 and VOICE 3 as well and add some chords to the music.
3. Improve the graphics to actually display the note being played instead of just the descriptive letter.

STOPWATCH

PURPOSE AND DISCUSSION

If you are only using your Commodore 64 for making calculations or other "normal" computer work, you are missing out on something. The Commodore 64 has a very accurate internal timer, which can be very useful. This progam uses it in a very obvious way—as a stopwatch. Using a computer as a stopwatch gives you the advantage of leaving the last few timings on the screen for reference while you are making the next timing. Of course, the computer is "smart" enough to allow you to get "lap" times as well as the final time.

HOW TO USE IT

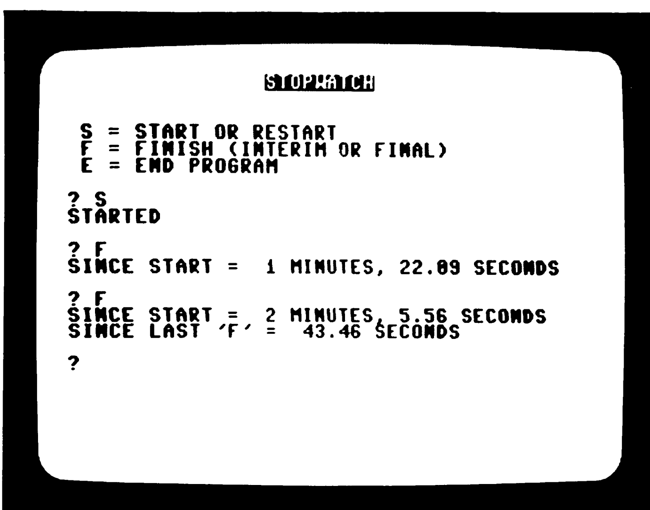
The opening messages from the program show you your two options. Pressing the **S** key causes the stopwatch to start (or restart, if you had already started it). Pressing the **F** key a second time causes the program to show you both the time since the start and the time since the last **F** was pressed. This lets you see interim or "lap" times. Pressing the **E** key ends the program.

For example, suppose you want to time a one-mile race that is run as four laps around a quarter-mile track. Before the race begins, start the program running with the RUN command. When the starting gun is fired to start the race, press the **S** key. At the end of the first lap around the track, press the **F** key. This causes the program to show the time since the race started. When the second lap is completed, press the **F** key again. The program will show the time of the second lap. Press the **F** key

again when the third and fourth laps are finished to get the time since start and lap times. Of course, the time since start at the end of the fourth lap is the final time of the race, even though the stopwatch keeps running. At that time, you can either press **S** to restart the stopwatch (when the next race begins) or press the **E** key to stop the program.

The internal timer of the Commodore 64 is accurate to one sixtieth (1/60) of a second. The program displays the time as though it is accurate to .01 seconds. As a result, the second decimal place is not precise. The actual time is within plus or minus .02 seconds of the time that is shown. Also, because of the computation time required to display each timing, wait at least .4 seconds between consecutive **F** suppressions.

SAMPLE RUN



First the operator presses S to start the stopwatch. After about one minute and 22 seconds, he or she presses F. After another 43.46 seconds, F is pressed again.

PROGRAM LISTING

```
READY.

100 REM: STOPWATCH
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 PRINT CHR$(147)
130 PRINT TAB(15);CHR$(18);"STOPWATCH":PRINT:PRINT
140 PRINT" S = START OR RESTART"
150 PRINT" F = FINISH (INTERIM OR FINAL)"
155 PRINT" E = END PROGRAM"
160 PRINT:PRINT"? ";
170 GET R$:IF R$="" THEN 170
180 W=TI:PRINT R$:S$ = R$
190 IF R$="S" THEN 300
200 IF R$="F" THEN 400
205 IF R$="E" THEN END
210 GOTO 170
300 L=W:S=W:PRINT"STARTED "
310 GOTO 160
400 J=W-S:GOSUB 500
410 PRINT"SINCE START = ";R$
420 IF S=L THEN L=W:GOTO 160
430 J=W-L:GOSUB 500
440 PRINT"SINCE LAST 'F' = ";R$
450 L=W:GOTO 160
500 J=J/60:J=INT(100*J):J=J/100:R$=""
510 IF J>60 THEN 530
520 R$=R$+STR$(J)+" SECONDS":RETURN
530 K=INT(J/60):J=J-(K*60):J=INT(100*J):J=J/100
540 R$=STR$(K)+" MINUTES,":GOTO 520

READY.
```

EASY CHANGES

1. To always display the time in seconds (instead of minutes and seconds when the time is greater than 60 seconds), delete line 510.
2. To eliminate blank lines on the screen (thus allowing more timings to be retained on the screen), eliminate the first PRINT and colon on line 160.
3. To display the time to the nearest tenth of a second (instead of the nearest hundredth), change the four constant values of 100 in lines 500 and 530 to 10.

MAIN ROUTINES

120-150	Displays the title and two options.
160-170	Displays question mark and waits for a key to be pressed.
180-210	Saves time of key suppression. Checks which key was pressed.
300-310	Saves starting time and displays STARTED.
400-450	Displays total and lap times.
500-540	Subroutine to convert "jiffies" in J into minutes and seconds in R$.

MAIN VARIABLES

R$	Keyboard character pressed by operator. Also used when displaying minutes and seconds as a string.
W	Work variable for saving current time.
L	Time that last **S** or **F** was pressed.
S	Starting time.
J	Elapsed time (in jiffies) to be displayed.
K	Number of minutes to be displayed when elapsed time is greater than 60 seconds.

SUGGESTED PROJECTS

1. Instead of displaying only the last lap time, display the last two or three lap times. Or, display *all* lap times since the start by saving each time in an array. Allow for at least twenty entries.
2. Allow for the possibility of the time automatically resetting to zero during a timing (by passing midnight or 24 hours since the Commodore 64 was turned on). Allow 24 hours worth of jiffies to W if W is less than S.

WEEKDAY

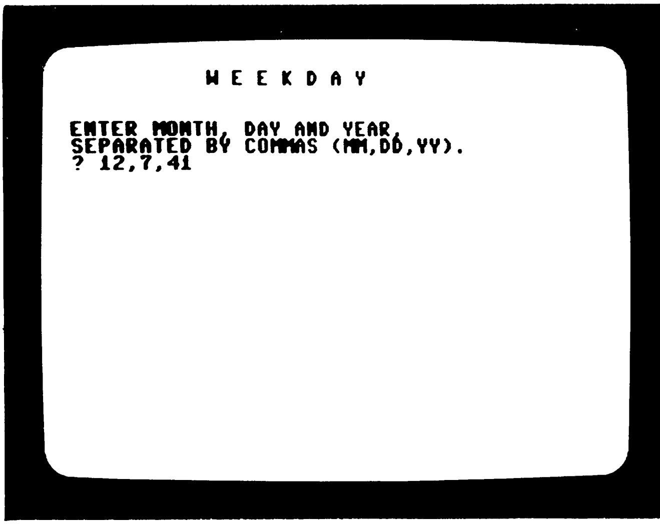
PURPOSE AND DISCUSSION

Weekday is a simple program that provides the day of the week for any date in the twentieth century. Its basis is an old parlor trick described by Scot Morris in the GAMES section of a 1981 issue of *OMNI* magazine.

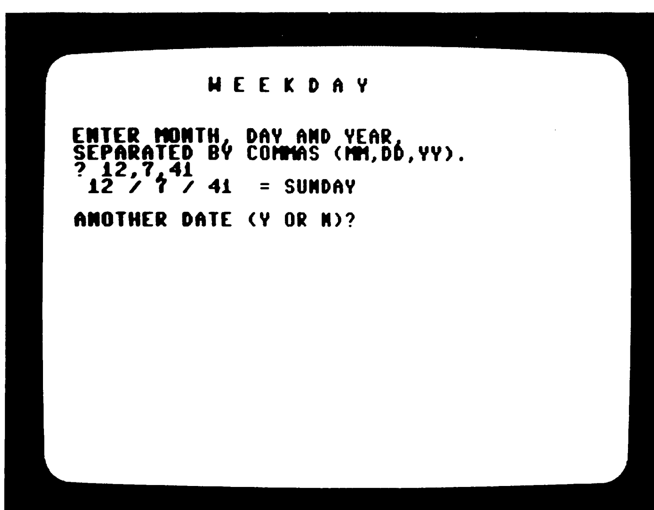
HOW TO USE IT

The program starts by displaying its title and asking for the month, day and year for which the conversion is desired. You enter the three items separated by commas and then press the **RETURN** key. The program then validates that a good date was entered and then calculates the day of the week and displays it. It then asks if you wish to enter another date or end the program. If you reply Y, it prompts you for another date. If you reply N, the program ends.

SAMPLE RUN



The program requests that a date be entered and the operator replies with December 7, 1941.



The program calculates that December 7, 1941 was a Sunday, then asks if another date is to be entered.

PROGRAM LISTING

```
READY.

100 REM: WEEKDAY
110 REM: COPYRIGHT 1983
111 REM: TOM RUGG, PHIL FELDMAN, AND
112 REM: WESTERN SYSTEMS GROUP
120 PRINT CHR$(147):CLR
130 DIM DW$(8),MI(13)
150 DW$(0)="SATURDAY ":DW$(1)="SUNDAY    "
160 DW$(2)="MONDAY    ":DW$(3)="TUESDAY   "
170 DW$(4)="WEDNESDAY":DW$(5)="THURSDAY "
180 DW$(6)="FRIDAY    "
190 MI(1)=1:MI(2)=4:MI(3)=4:MI(4)=0
200 MI(5)=2:MI(6)=5:MI(7)=0:MI(8)=3
210 MI(9)=6:MI(10)=1:MI(11)=4:MI(12)=6
300 PRINT TAB(10);"W E E K D A Y":PRINT
310 PRINT:PRINT"ENTER MONTH, DAY AND YEAR,"
320 PRINT"SEPARATED BY COMMAS (MM,DD,YY)."
330 INPUT MO,DA,YR
340 IF MO < 1 OR MO > 12 THEN 500
350 IF DA < 1 OR DA > 31 THEN 500
360 IF YR < 0 OR YR > 99 THEN 500
365 GOSUB 540
370 WO = (YR/4)+YR
380 WO = WO+MI(MO)+DA
390 WS = WO
400 WO = INT(WO/7)
410 WO = WS-(WO*7)
420 PRINT MO;"/";DA;"/";YR;" = ";DW$(WO):PRINT
430 INPUT"ANOTHER DATE (Y OR N)";C$
440 IF C$ = "N" THEN END
450 IF C$ = "Y" THEN 310
460 GOTO 430
500 PRINT"INPUT DATE IS INVALID"
510 END
540 IF MO = 4 AND DA > 30 THEN 500
550 IF MO = 6 AND DA > 30 THEN 500
560 IF MO = 9 AND DA > 30 THEN 500
570 IF MO = 11 AND DA > 30 THEN 500
590 IF MO <> 2 THEN RETURN
600 IF DA < 29 THEN RETURN
610 IF DA > 29 THEN 500
620 WY = INT(YR/4)
630 WY = YR - (WY*4)
640 IF WY <> 0 THEN 500
650 RETURN

READY.
```

EASY CHANGES

1. To have the program immediately request another date instead of asking if you want another, delete lines 430, 440 and 450. Also, change line 460 to:

```
460 GOTO 310
```

MAIN ROUTINES

120-210	Initializes variables and sets up arrays.
300	Displays title.
310-330	Gets month, day and year from operator.
340-365	Validates input date.
370-420	Computes and displays day of the week.
430-460	Prompts operator for option to end program or supply another date.
500-510	Issues error message and ends program, if input date is invalid.
540-650	Subroutine to validate date in February, allowing February 29 if it is a leap year.

MAIN VARIABLES

DW$	Array of valid days of the week.
MI	Array of numeric values used to adjust calculation depending on month.
MO	Month entered by operator.
DA	Day entered by operator.
YR	Year entered by operator.
WO	Work variable.
WS	Work variable.
C$	Input string from operator for continuation option.
WY	Work variable.

SUGGESTED PROJECTS

1. Modify the program to work for other centuries. Research formulas that are more precise and don't require adjustments.
2. Read about the differences between the JULIAN and GREGORIAN calendars. This program is based on the GREGORIAN calendar. Write one that uses the JULIAN calendar as its basis.

Bibliography

BOOKS

Bell, R. C., *Board and Table Games From Many Civilizations*, Oxford University Press, London, 1969. (WARI)

Cohen, Daniel, *Biorhythms in Your Life*, Fawcett Publications, Greenwich, Connecticut, 1976. (BIORHYTHM)

Commodore Business Machines, Inc., Wayne, PA., *Commodore 64 User's Guide*.

Crow, E. L., David, F. A., and Maxfield, M. W., *Statistics Manual*, Dover Publications, New York, 1960. (STATS)

Croxton, F. E., Crowden, D. J., and Klein, S., *Applied General Statistics* (Third Edition), Prentice-Hall, Englewood Cliffs, N.J., 1967. (STATS)

Elson, Louis C., *Elson's Pocket Music Dictionary*, Oliver Ditson Company, Bryn Mawr, Pennsylvania, 1909. (TUNE)

Gruenberger, Fred J., and Jaffray, George, *Problems for Computer Solution*, John Wiley and Sons, New York, 1965. (BIRTHDAY, PI, SORTLIST)

Gruenberger, Fred J., and McCracken, Daniel D., *Introduction to Electronic Computers*, John Wiley and Sons, New York, 1961. (MILEAGE, PI, PYTHAG, WARI as Oware)

Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

Knuth, Donald E., *The Art of Computer Programming (Volume 3)*, Addison-Wesley, Reading, Massachusetts, 1973. (SORTLIST)

Kuo, S. S., *Computer Applications of Numerical Methods*, Addison-Wesley, Reading, Massachusetts, 1972. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

McCracken, Daniel D., and Dorn, W. S., *Numerical Methods And FORTRAN Programming*, John Wiley and Sons, New York, 1964. (CURVE, DIFFEQN, INTEGRATE, SIMEQN)

Orr, William I., *Radio Handbook* (19th Edition), Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1972. (HAMCODE)

Shefter, Harry, *Faster Reading Self-Taught*, Washington Square Press, New York, 1960. (TACHIST)

PERIODICALS

Feldman, Phil, and Rugg, Tom, "Pass the Buck," *Kilobaud*, July 1977, pp. 90-96. (DECIDE)

Fliegel, H. F., and Van Flandern, T. C., "A Machine Algorithm for Processing Calendar Dates," *Communications of the ACM*, October, 1968, P. 657. (BIORHYTHM)

Errata Offer

All of the programs in this book have been tested carefully and are working correctly to the best of our knowledge. However, we take no responsibility for any losses which may be suffered as a result of errors or misuse. You must bear the responsibility of verifying each program's accuracy and applicability for your purposes.

If you want to get a copy of an errata sheet that lists corrections for any errors or ambiguities we have found to date, send one dollar ($1.00) and a self addressed stamped envelope (SASE) to the address below. Ask for errata for this book (by name). We hope we won't have any errors to tell you about, in which case we'll try to send you some other worthwhile information about the Commodore 64.

If you think you've found an error, please let us know. If you want an answer, include a SASE.

Please keep in mind that the most likely cause of a program working incorrectly is a typing error. Please check your typing *very* carefully before you send us an irate note about an error in one of the programs.

Western Systems Group
Errata—Commodore 64 Programs
P.O. Box 162896
Sacramento, CA 95816

COMMODORE USERS TAKE NOTE . . .

If you like *More Than 32 BASIC Programs for the Commodore 64 Computer*, you will appreciate having the programs on a cassette or diskette which is ready to run on your Commodore 64 computer. The software has a 'forever guarantee' (any problems, simply return the disk with $5 and we will send you a new one). Not only will it save your typing time, the cassette will save you time fretting about errors that are so easy to make. Interested?

☐ You bet I'm interested! Please send me the 5¼ diskette________ cassette________ for my Commodore 64.

☐ Please find my check in the amount of $19.95 and rush my Commodore 64 software to the address below.

☐ Please charge my ______VISA ______M/C and send to the address below.

ACCT #__________________ Exp. Date ____________

Signature ________________________________

Name ____________________________

Address ____________________________

City, State, Zip ________________________

☐ Please send me your catalog entitled Brain Food.

(To expedite your order, phone 1-800-547-1842 and charge to your VISA or M/C)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 143 BEAVERTON, OREGON

Postage will be paid by addressee

dilithium Software

P.O. Box 606
Beaverton, OR 97075

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 143 BEAVERTON, OREGON

Postage will be paid by addressee

dilithium Software

P.O. Box 606
Beaverton, OR 97075

If you are a Commodore user, this book is chock full of programs designed specifically for your machine. The book includes games, applications, educational programs, graphics, mathematics, and various miscellaneous programs. Each of the chapters fully documents each program by providing a complete source listing of the program, its purpose, and how to use it. The authors also tell you how to adapt the programs by making simple modifications.

The programs are fully tested and ready to run!

Software available in the book/software package of the same name.



0-88056-112-2